

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN**

NGUYỄN PHÚC KHANG

**XÂY DỰNG THỬ NGHIỆM Ví ĐIỆN TỬ
CHO TIỀN ẢO BITCOIN**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT
LỚP CỬ NHÂN TÀI NĂNG**

TP. Hồ Chí Minh, tháng 07/2021

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN**

NGUYỄN PHÚC KHANG – 1712224

**XÂY DỰNG THỬ NGHIỆM Ví ĐIỆN TỬ
CHO TIỀN ẢO BITCOIN**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT
LỚP CỬ NHÂN TÀI NĂNG**

GIÁO VIÊN HƯỚNG DẪN

TS. Trương Toàn Thịnh

TP. Hồ Chí Minh, tháng 07/2021

Lời cảm ơn

Em xin chân thành cảm ơn khoa Công nghệ Thông tin, trường Đại học Khoa học Tự nhiên, ĐHQG-HCM đã tạo điều kiện rất tốt cho em thực hiện đề tài này. Khóa luận này đã không thể thành công nếu không có sự động viên, hỗ trợ và cổ vũ từ gia đình, bạn bè, giáo viên hướng dẫn và các anh chị đi trước.

Em xin cảm ơn thầy Trương Toàn Thịnh, đã đồng ý hướng dẫn khóa luận cho em, và cảm ơn thầy đã nhiệt tình trao đổi, đồng thời chia sẻ nhiều tài liệu, kiến thức, giúp em hoàn thành khóa luận này một cách tốt đẹp nhất có thể. Em cũng muốn gửi lời cảm ơn tới thầy Trần Minh Triết, thầy Phạm Nguyễn Sơn Tùng, và thầy Hồ Tuấn Thanh vì đã kéo em lên trong những lúc em gục ngã, đã là điểm tựa, và động viên, giúp đỡ em trong những giai đoạn trầm lắng của bốn năm đại học, đã rèn giũa em trở thành một con người có trách nhiệm, có kỹ năng, và có kiến thức.

Cuối cùng, em muốn gửi lời cảm ơn chân thành nhất đến với bố, mẹ em vì tình yêu vô bờ bến của họ; những người bạn, người anh, người chị đã luôn ủng hộ, giúp đỡ và động viên em trong quá trình thực hiện khóa luận này.

Hồ Chí Minh, tháng 07/2021

Nguyễn Phúc Khang

Đề cương chi tiết

Tên đề tài: Xây dựng thử nghiệm ví điện tử cho tiền ảo Bitcoin.
Giáo viên hướng dẫn: TS. Trương Toàn Thịnh
Sinh viên thực hiện: Nguyễn Phúc Khang (1712224)
Thời gian thực hiện: 01/03/2021 – 14/07/2021
Loại đề tài: Nghiên cứu - Ứng dụng
Nội dung đề tài: <p>Hiện nay, tiền ảo là một công nghệ mới và ngày càng thu hút sự chú ý của nhiều người. Mục tiêu của đề tài là xây dựng thử nghiệm một ví điện tử dùng cho tiền ảo bitcoin, đồng tiền điện tử phổ biến nhất hiện tại, với mục đích quản lý một ví bitcoin đơn giản, hỗ trợ việc tạo giao dịch và quản lý những vấn đề hóc búa thay cho người dùng.</p> <p>Nội dung chi tiết của đề tài bao gồm:</p> <ul style="list-style-type: none">• Tìm hiểu về bitcoin và cách thức hoạt động.• Tìm hiểu về Bitcoin Core – một mẫu cài đặt của nút đầy đủ cho hệ thống mạng bitcoin.• Tìm hiểu về địa chỉ và khóa – hai thành phần không thể thiếu trong một giao dịch bitcoin.• Tìm hiểu về các cấu trúc ví: ví ngẫu nhiên, ví xác định, và ví HD. Đánh giá những ưu điểm, nhược điểm của các loại ví và chọn ra cấu trúc ví để sử dụng.• Tìm hiểu về giao dịch bitcoin, cấu trúc một giao dịch, và những vấn đề liên quan: chữ ký điện tử, cách ký một giao dịch, phí giao dịch, ...

- Nghiên cứu các thư viện hỗ trợ cài đặt các thao tác của ví bitcoin.
- Tiến hành cài đặt ví và các thao tác giao dịch tiêu chuẩn của bitcoin.
- Thử nghiệm chương trình đã cài đặt trên hệ thống testnet bitcoin.

Kế hoạch thực hiện:

Giai đoạn 1 – Tìm hiểu về bitcoin và thử nghiệm bitcoin core

- 01/03/2021 – 15/03/2021: Tìm hiểu những thành phần cơ bản và cách thức hoạt động của bitcoin.
- 16/03/2021 – 25/03/2021: Tìm hiểu cách thức cài đặt và cách sử dụng Bitcoin Core – cài đặt mẫu của một nút bitcoin đầy đủ.

Giai đoạn 2 – Tìm hiểu về các thành phần của một giao dịch bitcoin, các cấu trúc ví và cách cấu tạo giao dịch

- 26/03/2021 – 10/04/2021: Tìm hiểu và cài đặt các bộ công cụ hỗ trợ lập trình bitcoin.
- 11/04/2021 – 15/04/2021: Tìm hiểu về khóa và địa chỉ bitcoin.
- 16/04/2021 – 20/04/2021: Tìm hiểu về các cấu trúc ví cho bitcoin.
- 21/04/2021 – 15/05/2021: Cài đặt các hàm sinh khóa, địa chỉ và ví HD dùng cho bitcoin.
- 16/05/2021 – 05/06/2021: Tìm hiểu về các thành phần và thao tác liên quan đến một giao dịch bitcoin.
- 06/06/2021 – 14/06/2021: Cài đặt giao dịch bitcoin.
- 15/06/2021 – 18/06/2021: Tiến hành thử nghiệm và hoàn thiện các thao tác giao dịch.

Giai đoạn 3 – Tiến hành cài đặt giao diện tương tác dòng lệnh và các hàm hỗ trợ người dùng

- 19/06/2021 – 26/06/2021: Tìm hiểu các API hỗ trợ truy vấn dữ liệu bitcoin.

- 27/06/2021 – 05/07/2021: Cài đặt giao diện tương tác dòng lệnh
- 06/07/2021 – 14/07/2021: Viết báo cáo và hoàn thiện khóa luận.

Xác nhận của giáo viên hướng dẫn

Ngày tháng năm

Sinh viên thực hiện

TS. Trương Toàn Thịnh

Nguyễn Phúc Khang

Mục lục

Lời cảm ơn	i
Đề cương chi tiết.....	ii
Mục lục	v
Danh mục hình ảnh.....	ix
Danh mục bảng.....	xi
Tóm tắt khóa luận	xii
Chương 1 Mở đầu.....	1
1.1. Giới thiệu tổng quan	1
1.2. Lý do thực hiện đề tài	2
1.3. Mục tiêu đề tài	3
1.4. Nội dung đề tài	3
1.5. Phạm vi đề tài	4
Chương 2 Sơ lược về bitcoin	5
2.1. Lịch sử hình thành	5
2.2. Nền tảng mật mã.....	6
2.2.1. Thuật toán mã hóa.....	6
2.2.2. Hàm băm mật mã.....	7
2.2.3. Chữ ký điện tử	8
2.3. Sơ lược cách thức hoạt động	8
2.3.1. Tình huống.....	8
2.3.2. Cách hoạt động của mạng lưới bitcoin	9
2.4. Tính an toàn và chống gian lận.....	11

2.4.1.	Sự an toàn của một giao dịch bitcoin.....	11
2.4.2.	Khả năng chống gian lận của blockchain	11
Chương 3	Bitcoin Core và libbitcoin	13
3.1.	Giới thiệu về Bitcoin Core.....	13
3.2.	Cài đặt Bitcoin Core	14
3.2.1.	Bản cài đặt do nhà phát triển đóng gói	14
3.2.2.	Tự biên dịch và cài đặt.....	15
3.3.	Sử dụng Bitcoin Core	19
3.3.1.	Cấu hình Bitcoin Core	19
3.3.2.	Lần đầu khởi động Bitcoin Core.....	20
3.3.3.	Giao tiếp với Bitcoin Core.....	20
3.4.	Hệ thống thư viện libbitcoin.....	23
3.4.1.	Giới thiệu về libbitcoin	23
3.4.2.	Cách cài đặt và sử dụng libbitcoin-system	24
Chương 4	Khóa và địa chỉ	28
4.1.	Nền tảng mật mã.....	28
4.1.1.	Sơ lược mã hóa công khai.....	29
4.1.2.	Thuật toán mã hóa đường cong elliptic	29
4.2.	Khóa.....	32
4.2.1.	Khóa bí mật.....	32
4.2.2.	Khóa công khai	33
4.3.	Địa chỉ.....	34
4.3.1.	Định dạng mã hóa Base58 và Base58Check	35
4.3.2.	Định dạng khóa.....	37

4.4. Cài đặt khóa và địa chỉ sử dụng libbitcoin	39
Chương 5 Ví bitcoin	42
5.1. Tổng quan các công nghệ ví.....	42
5.1.1. Ví ngẫu nhiên.....	42
5.1.2. Ví xác định.....	43
5.1.3. Ví HD.....	44
5.2. Chi tiết về ví HD.....	44
5.2.1. Mã mnemonic	45
5.2.2. Dẫn xuất khóa	48
5.2.3. Đường dẫn khóa.....	51
Chương 6 Giao dịch bitcoin.....	53
6.1. Tổng quan về giao dịch bitcoin	53
6.2. Đầu vào và đầu ra giao dịch	53
6.2.1. Đầu ra giao dịch.....	54
6.2.2. Đầu vào giao dịch	57
6.2.3. Xâu chuỗi giao dịch	59
6.3. Cách thực thi script và loại script phổ thông.....	60
6.3.1. Script và cách thức thực thi	60
6.3.2. Pay-to-Public-Key-Hash script.....	62
6.4. Chữ ký điện tử	63
6.4.1. Thực hiện ký	64
6.4.2. Xâu chuỗi chữ ký (DER)	64
6.4.3. Các loại chữ ký	65
6.4.4. Ký giao dịch bao gồm nhiều đầu vào	66

6.5.	Một số thông tin thêm về giao dịch	67
6.5.1.	Phí giao dịch	67
6.5.2.	Nhúng nội dung vào giao dịch.....	67
6.5.3.	Phát giao dịch lên mạng lưới bitcoin	68
Chương 7	Một số tính năng nhỏ và giao diện tương tác trên dòng lệnh.....	69
7.1.	Các tính năng hỗ trợ	69
7.1.1.	Chain.so – API hỗ trợ truy vấn trên blockchain	69
7.1.2.	Một số tính năng hỗ trợ.....	71
7.2.	Giao diện tương tác trên dòng lệnh	73
7.2.1.	Giao diện chính.....	73
7.2.2.	Tạo giao dịch	74
7.2.3.	Hạn chế và thiếu sót.....	76
Chương 8	Thử nghiệm chương trình	78
8.1.	Thử nghiệm lý thuyết	78
8.2.	Thử nghiệm trên mạng testnet bitcoin.....	79
Chương 9	Kết luận và định hướng phát triển	81
9.1.	Kết luận.....	81
9.2.	Định hướng phát triển.....	81
Tài liệu tham khảo	83

Danh mục hình ảnh

Hình 1.1: Logo của đồng bitcoin, đồng tiền mã hóa phân tán đầu tiên	1
Hình 2.1: Sử dụng công cụ blockchain.com để xem thông tin giao dịch	9
Hình 2.2: Sự liên kết giữa các block với nhau	10
Hình 2.3: Hình minh họa cho việc “chạy đua” giữa hai nhánh blockchain.....	12
Hình 3.1: Trang tải về tập tin cài đặt Bitcoin Core	14
Hình 3.2: Giao diện ứng dụng Bitcoin Core	15
Hình 3.3: Xem những hàm Bitcoin Core hỗ trợ.....	21
Hình 3.4: Ví dụ xem hỗ trợ hàm <i>getrawtransaction</i>	22
Hình 3.5: Lấy nội dung giao dịch	22
Hình 3.6: Giải mã nội dung giao dịch	23
Hình 3.7: Sơ đồ quan hệ giữa các thư viện thuộc hệ thống libbitcoin.....	24
Hình 3.8: Nội dung được xuất ra khi cài đặt thành công libbitcoin-system	27
Hình 4.1: Hình ảnh đường cong elliptic trên trường số thực	30
Hình 4.2: Hình ảnh đường cong elliptic với $p = 17$	30
Hình 4.3: Hình minh họa tổng của hai điểm trên đường cong elliptic	31
Hình 4.4: Mối quan hệ giữa khóa bí mật, khóa công khai, và địa chỉ Bitcoin	32
Hình 4.5: Tạo khóa bí mật sử dụng Bitcoin Core	33
Hình 4.6: Quá trình tạo địa chỉ bitcoin.....	34
Hình 4.7: Quy trình mã hóa Base58Check	36
Hình 4.8: Mã hóa Base58Check cho khóa bí mật với định dạng WIF nén	37
Hình 4.9: Kết quả thực thi chương trình mẫu sinh khóa và địa chỉ	41
Hình 5.1: Hình mô phỏng ví ngẫu nhiên.....	43
Hình 5.2: Hình mô phỏng ví xác định.....	43
Hình 5.3: Hình mô phỏng ví HD.....	44
Hình 5.4: Quy trình tạo mã mnemonic từ entropy	46
Hình 5.5: Quy trình tạo seed từ mã mnemonic	47

Hình 5.6: Quy trình tạo khóa gốc từ seed	48
Hình 5.7: Quy trình dẫn xuất khóa bí mật của ví HD	49
Hình 5.8: Quy trình dẫn xuất khóa công khai của ví HD	50
Hình 6.1: Quy trình thực thi của script	61
Hình 7.1: Giao diện chính của trang chain.so	70
Hình 7.2: Ví dụ thực hiện lời gọi lấy thông tin địa chỉ trên testnet	71
Hình 7.3: Ví dụ lời gọi lấy thông tin giao dịch	72
Hình 7.4: Ví dụ lời gọi lấy thông tin UTXO của địa chỉ	72
Hình 7.5: Hướng dẫn khởi động chương trình.....	73
Hình 7.6: Giao diện tương tác dòng lệnh	73
Hình 7.7: Các lệnh chương trình hỗ trợ	74
Hình 7.8: Quy trình vào chế độ tạo giao dịch	75
Hình 7.9: Ví dụ quá trình tạo giao dịch.....	76
Hình 8.1: Thử nghiệm quy trình tạo khóa và địa chỉ bitcoin	78
Hình 8.2: Thử nghiệm quy trình tạo ví HD và master key	79
Hình 8.3: Thử nghiệm quy trình dẫn xuất xóa với các bộ kiểm thử của BIP-0032..	79
Hình 8.4: Thử nghiệm quy trình cấu tạo giao dịch	79
Hình 8.5: Thử nghiệm tạo giao dịch có nhiều đầu vào và đầu ra thành công.....	80

Danh mục bảng

Bảng 3.1: Bảng tóm tắt các thư viện hỗ trợ quá trình biên dịch Bitcoin Core.....	16
Bảng 3.2: Bảng tóm tắt các thư viện hỗ trợ quá trình biên dịch libbitcoin-system ..	25
Bảng 4.1: Một số loại dữ liệu áp dụng Base58	36
Bảng 5.1: Độ lớn của các thành phần trong quá trình tạo mã mnemonic	46
Bảng 5.2: Một số ví dụ về đường dẫn khóa	51
Bảng 6.1: Định dạng chuỗi của một đầu ra giao dịch	56
Bảng 6.2: Giải mã đầu ra giao dịch mẫu	57
Bảng 6.3: Định dạng chuỗi của đầu ra giao dịch	58
Bảng 6.4: Giải mã đầu vào giao dịch mẫu	59
Bảng 6.5: Giải thích phiên bản và thời gian khóa trong giao dịch	59
Bảng 6.6: Các thành phần cần có trong một chuỗi giao dịch.....	60
Bảng 6.7: Loại chữ ký và giá trị cờ hiệu tương ứng	65

Tóm tắt khóa luận

Tiền ảo, hay cụ thể hơn trong đề tài này là tiền mã hóa, đang là một trong những vấn đề được bàn tán sôi nổi kể từ khi đồng tiền mã hóa phân tán đầu tiên, bitcoin, bắt đầu trở nên phổ biến vào đầu những năm 2010. Gần đây, mọi ánh mắt đều tiếp tục hướng về đồng tiền ảo này và cơn sốt bitcoin lại một lần nữa được đẩy lên khi bitcoin đạt đỉnh điểm giá trị một đồng tương ứng với khoảng 63,314 Đô la Mỹ vào cuối quý 1, đầu quý 2 năm 2021. Sự kiện bitcoin đạt ngưỡng giá trị này cũng đã kéo theo một làn sóng đẩy giá trị của các đồng tiền ảo khác lên mức cao chưa từng thấy (ví dụ như Ethereum, Dodecoin, Solana, ...). Với sự gia tăng của cơn sốt tiền ảo, nhiều người đã và đang cố gắng sở hữu tiền ảo nhiều hơn bao giờ hết. Chính phủ của các quốc gia cũng vì thế mà đã bắt đầu tìm hiểu, nghiên cứu để đưa ra những dự luật cho loại tiền tệ ảo này. Tiền ảo không chỉ đơn thuần là bitcoin, mà nó còn nói đến nhiều đồng tiền khác với số lượng rất lớn. Càng ngày càng có nhiều đồng tiền ảo được sinh ra và đưa vào thử nghiệm, được liệt kê lên những sàn giao dịch lớn (ví dụ như Binance, Coinbase, Liquid, ...) dẫn tới việc người dùng muốn quản lý loại tài sản số này ngày càng khó khăn.

Đề tài được sinh ra để cố gắng giải quyết vấn đề này. Mục tiêu đề tài như là một bước đầu tạo ra nền móng để có thể phát triển lên một hệ thống ví điện tử giúp người dùng hệ thống toàn bộ các loại tiền ảo phổ biến vào trong một ứng dụng duy nhất. Trọng tâm của đề tài này sẽ là việc cài đặt các thao tác xây dựng ví HD và thao tác cấu tạo giao dịch, nhằm tạo ra một hệ thống nền tảng, có thể mở rộng ra ngoài bitcoin, nhằm cho người dùng có một sự kiểm soát toàn bộ số tài sản ảo của họ một cách tổng quan, ít phải suy nghĩ cho những người dùng cơ bản, cho tới việc kiểm soát một cách cẩn kẽ và chi tiết cho những người dùng nâng cao hơn.

Chương 1

Mở đầu

Chương này sẽ giới thiệu tổng quan về đề tài, lý do thực hiện đề tài để từ đó đưa ra được mục tiêu của khóa luận. Bên cạnh đó, nội dung tóm tắt khóa luận cũng được trình bày ở cuối chương.

1.1. Giới thiệu tổng quan

Tiền ảo (hay *tiền điện tử*, *tiền số*) là một dạng tài sản điện tử được thiết kế để sử dụng như một phương tiện trao đổi với sự sở hữu các “đồng tiền” của một người được ghi chép bởi một cuốn sổ cái điện tử. Cuốn sổ cái này sử dụng các kỹ thuật bảo mật mạnh mẽ nhằm giữ tính an toàn cho các giao dịch, sinh thêm tiền ảo, và cuối cùng là dùng để xác minh giao dịch.



Hình 1.1: Logo của đồng bitcoin, đồng tiền mã hóa phân tán đầu tiên

Bitcoin được sinh ra vào năm 2009 bởi một người hoặc nhóm người với tên Satoshi Sakamoto [1], là đồng tiền tiền ảo có ứng dụng mã hóa (tiền mã hóa), sử dụng mạng lưới phân tán đầu tiên được sinh ra đời, và cũng là đồng tiền ảo phổ biến nhất hiện tại. Sau đó không lâu, đồng tiền mã hóa thứ hai và cũng là một trong những đồng

phổ biến hiện nay, Ethereum, được sinh ra đời. Từ đó tới nay, số lượng đồng tiền ảo nói chung và tiền mã hóa nói riêng được sinh ra ngày càng nhiều.

Ngày nay, sự phổ biến của tiền ảo là một điều không thể chối cãi, đặc biệt là với sự phát triển không ngừng của các loại tiền mã hóa này. Trước khi bitcoin được sinh ra, mọi giao dịch ngân hàng đều được ghi chép lại trong cơ sở dữ liệu ngân hàng, là một hệ thống tập trung, dẫn tới những nguy cơ bị lộ dữ liệu, hay những bản ghi chép giao dịch có thể bị thay đổi mà không một ai có thể biết được là đã bị thay đổi. Tiền mã hóa giải quyết được hai vấn đề này, chúng đảm bảo được tính bảo mật cho người dùng bởi vì hệ thống mạng của các đồng tiền mã hóa được xây dựng xung quanh những kỹ thuật, tính chất của mật mã, giúp cho người dùng an toàn về danh tính.

Lý do tiếp theo vì sao tiền mã hóa đang dần trở nên phổ biến là vì mạng lưới của các loại đồng tiền này là một mạng lưới phân tán, tức không một ai là chủ sở hữu của cuốn sổ cái, mà chúng sẽ được phân tán và đối chứng, cập nhật lẫn nhau, dẫn tới nhiều phiên bản sổ cái giống hệt được nhiều người nắm giữ, tránh được kẻ gian lợi dụng để chuộc lợi cho bản thân, và người dùng không cần phải tin tưởng những trung tâm tài chính như ngân hàng, các công ty tài chính thế giới (VISA, Mastercard, ...) để phân phối và quyết định sự hợp lệ của giao dịch. Giải thích cho những lý do này sẽ được trình bày cụ thể hơn ở những chương sau.

1.2. Lý do thực hiện đề tài

Với việc tiền ảo ngày nay đang ngày càng trở nên phổ biến, số lượng người sở hữu cũng theo đó mà tăng lên. Việc quản lý rất nhiều loại tiền ảo khác nhau qua nhiều ứng dụng ví điện tử làm người dùng là một điều hết sức khó khăn và cũng không kém phần hóc búa.

Một trong những vấn đề với các loại ví điện tử thông dụng cho các loại tiền ảo này thường chưa thực sự cho phép người dùng có kiểm soát số dư của mình một cách tự chủ nhất. Với tính chất của các loại tiền ảo, để thực hiện các giao dịch “chuyển tiền”, mình cần phải có một cơ chế để xác minh rằng mình hiện tại là chủ sở hữu của khoản tài sản này, và sau đó thì mình mới có khả năng dùng khoản tài sản đó và thực

hiện giao dịch. Các ví điện tử thông dụng hiện tại vẫn chưa cho phép người dùng sử dụng khả năng này với mong muốn cá nhân của họ.

Vì lý do trên, đề tài này được sinh ra với mong muốn nghiên cứu, và xây dựng một ví điện tử cơ bản, là nền tảng nhất để các thế hệ sau này có thể sử dụng và phát triển tiếp tục, hỗ trợ thêm nhiều loại đồng tiền ảo phổ biến khác như Ethereum hay Dogecoin.

1.3. Mục tiêu đề tài

Mục tiêu cuối cùng của đề tài là *nghiên cứu và xây dựng* thử nghiệm một ví điện tử dành cho đồng tiền ảo bitcoin hiện tại, tập trung chủ yếu vào hai thành phần chính đó là *cấu trúc ví HD* và *cấu tạo giao dịch*.

1.4. Nội dung đề tài

Khóa luận bao gồm 9 chương, nội dung chính từng chương như sau:

- Chương 1: Trình bày bài toán đặt ra, xây dựng một ví điện tử cho tiền ảo bitcoin, và giới thiệu tổng quan về tiền ảo.
- Chương 2: Sơ lược về lịch sử bitcoin và cách thức hoạt động.
- Chương 3: Chương này sẽ giúp chúng ta có một cái nhìn sơ lược về cài đặt mẫu của một nút bitcoin đầy đủ.
- Chương 4: Chương sẽ đi vào tìm hiểu về hai thành phần quan trọng của một giao dịch bitcoin là khóa và địa chỉ.
- Chương 5: Một người dùng có thể sử dụng nhiều địa chỉ để nhận và gửi tiền, chương này sẽ trình bày ba loại công nghệ ví được sử dụng.
- Chương 6: Các thành phần trong một giao dịch bitcoin sẽ được tìm hiểu trong phần này, kèm theo đó là cách tạo một giao dịch bitcoin.
- Chương 7: Chương này sẽ đi qua về những tính năng bổ sung cho ứng dụng, những tính năng tuy nhỏ nhưng cũng không kém phần quan trọng.

- Chương 8: Chương này sẽ tập trung vào việc thử nghiệm chương trình trên mạng lưới kiểm thử (testnet) của bitcoin.
- Chương 9: Chương này sẽ trình bày về kết quả đã đạt được và hướng phát triển trong tương lai của đề tài.

1.5. Phạm vi đề tài

Đề tài tập trung vào việc phát triển một dạng ví điện tử cơ bản nhất, giúp người dùng có thể dễ dàng thao tác trên các tài khoản bitcoin của mình. Chương trình sẽ giúp người dùng quản lý các khoản về địa chỉ nhận, cấu hình giao dịch và truy vấn số dư tài khoản. Đối tượng hướng tới của đề tài là những người đã, đang, và sẽ sở hữu tài sản dưới dạng tiền ảo bitcoin.

Chương 2

Sơ lược về bitcoin

Trước khi đi vào tìm hiểu các thành phần của bitcoin, chương này trình bày về lịch sử hình thành của đồng tiền ảo bitcoin, và điếm qua những kiến thức nền tảng xây dựng lên hệ thống bitcoin. Bên cạnh đó, chương cũng sẽ trình bày sơ lược về cách thức hoạt động của mạng lưới tiền ảo bitcoin, và cuối cùng là nhận xét về tính an toàn của hệ thống.

2.1. Lịch sử hình thành

Bitcoin (ký hiệu tiền tệ *BTC*) là một dạng tiền ảo phân tán, không phụ thuộc vào một ngân hàng trung tâm, hay một cá thể quản lý nào cả. Bitcoin được sinh ra vào năm 2008 khi sách trắng có tên “Bitcoin: A Peer-to-Peer Electronic Cash System” được công bố bởi Satoshi Nakamoto [2]. Năm 2009, đồng tiền ảo này bắt đầu được sử dụng rộng rãi khi cách thức hoạt động của đồng tiền được công bố dưới dạng mã nguồn mở có tên là Bitcoin Core [3].

Theo như sách trắng bitcoin, bitcoin là một dạng tiền ảo *phân tán*, hoạt động hoàn toàn dựa trên *mạng ngang hàng* (*Peer-to-Peer network*). Khi người dùng sử dụng các hệ thống tiền điện tử truyền thống của các ngân hàng hay tập đoàn, người dùng cần phải đặt niềm tin rằng họ sẽ giữ được tính bảo mật của giao dịch, và ngăn chặn sự trùng chi. Bitcoin đưa ra giải pháp cho vấn đề này bằng cách sử dụng ghi nhận thời gian lên cho giao dịch bằng cách gom các giao dịch lại thành một khối, sau đó thực hiện hàm băm cho khối và nối dài chúng cho những khối phía sau.

Satoshi Nakamoto hiện nay đã không còn hoạt động, không ai biết được người đứng sau cái tên này thực sự là ai cả. Tuy nhiên, mã nguồn và mạng lưới bitcoin hiện vẫn đang được tiếp tục phát triển bởi một nhóm các tình nguyện viên.

2.2. Nền tảng mật mã

Để hiểu được cách thức hoạt động của bitcoin, đầu tiên chúng ta cần nắm một số khái niệm về mật mã. Mật mã là một nhánh nghiên cứu giúp các thông tin được gửi đi một cách an toàn.

Trong bitcoin, mã hóa được sử dụng để giải quyết ba bài toán sau trong bốn bài toán mà mật mã giúp giải quyết¹: *chứng thực*, *tính toàn vẹn dữ liệu*, và *chống thoái thác trách nhiệm* [3]. Ba bài toán này được giải thích như sau:

- *Chứng thực*: Người nhận bản tin sẽ biết rằng ai thực sự là người gửi bản tin đó, không một ai khác có thể giả mạo danh tính người gửi thực sự;
- *Tính toàn vẹn dữ liệu*: Người nhận bản tin sẽ biết được bản tin này chưa bị thay đổi dưới hình thức nào cả, đồng thời không một ai khác có thể thay đổi bản tin gốc;
- *Chống thoái thác trách nhiệm*: Người gửi bản tin sẽ không thể phủ nhận rằng mình không là người gửi bản tin.

2.2.1. Thuật toán mã hóa

Mã hóa là một trong những phương pháp phổ biến để bảo mật dữ liệu trao đổi giữa hai bên. Nếu không có khóa để giải mã các bản tin này thì một bên thứ ba không thể nào giải mã và đọc được chúng.

Mã hóa có hai dạng lớn: *mã hóa đối xứng* và *mã hóa bất đối xứng* (hay còn một tên gọi khác là *mã hóa công khai*).

- Mã hóa đối xứng là dạng mã hóa khi khóa dùng để mã hóa cũng là khóa để giải mã. Với dạng mã hóa này, hai bên đều phải giữ bí mật về khóa, nếu khóa bị rò rỉ thì ai cũng có thể đọc được những tin đã bị mã hóa.

¹ Bốn bài toán mật mã giúp giải quyết: giữ bí mật dữ liệu (confidentiality), tính toàn vẹn dữ liệu (integrity), chứng thực (authentication), và chống thoái thác trách nhiệm (non-repudiation).

- Mã hóa bất đối xứng là dạng mã hóa mà khóa dùng để giải mã khác với khóa dùng để mã hóa. Hai thành phần của cặp khóa này có tên là *khóa bí mật* và *khóa công khai*.

Khóa bí mật, đúng như tên của nó, là khóa mà chỉ có duy nhất người tạo khóa giữ, không ai có quyền sở hữu khóa này ngoài người tạo khóa. Ngược lại với khóa bí mật, khóa công khai sẽ được công bố một cách rộng rãi với mọi người, và ai cũng có thể dùng khóa này.

Mã hóa bất đối xứng được thiết kế với mong muốn đạt được là chỉ có duy nhất một mình khóa bí mật mới có thể giải mã được một bản tin đã được mã hóa bởi khóa công khai tương ứng.

Thuật toán mã hóa giúp giải quyết bài toán: giữ bí mật dữ liệu.

2.2.2. Hàm băm mật mã

Trong bitcoin, hàm băm mật mã được sử dụng để đảm bảo tính toàn vẹn dữ liệu.

Hàm băm mật mã là một thuật toán có đầu vào là một bản tin, từ đó cho ra một bản tin mới có độ dài cố định, gọi là giá trị băm.

Hàm băm mật mã có một số tính chất sau đây:

- *Không thể đảo ngược* (còn được gọi là *tính một chiều*, hay *hàm một chiều*): Nếu với các thuật toán mã hóa, ta có thể dùng bản tin đã được mã hóa, đi ngược lại để thấy được bản tin gốc, thì với hàm băm, điều này là không thể. Ta có thể dễ dàng sinh ra giá trị băm; nhưng nếu chúng ta có giá trị băm thì ta không thể (hoặc rất khó) có lại bản tin gốc ban đầu.
- *Tính lãn cầu tuyết*: Trong bản tin gốc, ta chỉ cần thay đổi một bit duy nhất thì thuật toán sẽ sinh ra một giá trị băm hoàn toàn khác.
- *Tính xác định*: Một bản tin chỉ có một giá trị băm duy nhất, cho dù áp dụng hàm băm rất nhiều lần.
- *Tính chống đụng độ*: Hàm băm phải hạn chế trường hợp hai bản tin khác nhau có chung một giá trị băm.

- *Không thể dự đoán*: Ta không thể dự đoán trước được giá trị băm của một bản tin dưới bất kỳ hình thức nào.

Với các tính chất này, hàm băm giúp chúng ta giải quyết các bài toán: tính toàn vẹn dữ liệu, chứng thực, và chống thoái thác trách nhiệm.

2.2.3. Chữ ký điện tử

Chữ ký điện tử bao gồm hai phần chính:

- Ký: quy trình thực hiện việc “ký” bản tin, xác minh mình là người tạo ra bản tin tương ứng, sử dụng khóa bí mật trong mã hóa công khai;
- Xác minh chữ ký: bất kỳ ai cũng có thể sử dụng quy trình này cùng với khóa công khai của người ký để xác minh chữ ký.

Chữ ký điện tử giúp giải quyết các bài toán bảo mật sau: tính toàn vẹn dữ liệu, chứng thực, và chống thoái thác trách nhiệm.

2.3. Sơ lược cách thức hoạt động

2.3.1. Tình huống

Giả sử trong tài khoản của em hiện đang có 0.001 BTC, em muốn đi ra ngoài một cửa hàng cửa hàng tiện lợi và mua một món đồ trị giá 0.00015 BTC. Biết địa chỉ ví bitcoin của cửa hàng tiện lợi là `mmpx1rZE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ`, em có thể cấu tạo giao dịch tới địa chỉ ví của cửa hàng với số tiền mong muốn một cách tương đối đơn giản với sự giúp đỡ một ví điện tử. Sau khi giao dịch được tạo thành, ví sẽ giúp em phát giao dịch này lên hệ thống mạng bitcoin.

Chi tiết về cấu tạo giao dịch sẽ được trình bày chi tiết hơn ở Chương 6. Sau khi giao dịch đã được phát thành công, ai cũng có thể xem qua giao dịch trên blockchain bằng một công cụ khám phá blockchain.

Explorer > Bitcoin Testnet > Transaction

Search your transaction, an address or a block

USD

Summary

This transaction was first broadcast to the Bitcoin network on July 09, 2021 at 4:13 PM GMT+7. The transaction currently has 2 confirmations on the network. At the time of this transaction, 0.00099500 BTC was sent with a value of \$32.71. The current value of this transaction is now \$32.56. Learn more about [how transactions work](#).

Hash	938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a298...	2021-07-09 16:13
	mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epy 0.00100000 BTC	mmpx1rzE7Y6DF1ovBMRLTz4BQ6WkXXyECZ 0.00015000 BTC
		mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epy 0.00084500 BTC
Fee	0.00000500 BTC (2.212 sat/B - 0.553 sat/WU - 226 bytes)	0.00099500 BTC
		2 Confirmations

Hình 2.1: Sử dụng công cụ blockchain.com để xem thông tin giao dịch

(Nguồn: [https://www.blockchain.com/btc-](https://www.blockchain.com/btc-testnet/tx/938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a29899383d314924)

[testnet/tx/938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a29899383d314924](https://www.blockchain.com/btc-testnet/tx/938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a29899383d314924))

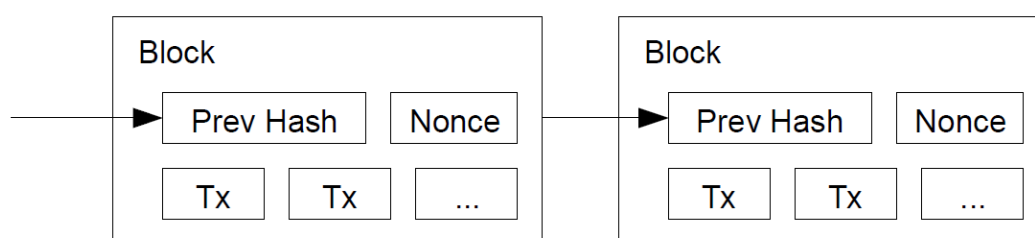
2.3.2. Cách hoạt động của mạng lưới bitcoin

Sau khi giao dịch mua đồ bên trên được phát lên mạng lưới, giao dịch này sẽ được ghi lại vào trong một danh sách chờ, hay còn được gọi là một khối (*block*). Sau khi block đầy, người xây dựng khối sẽ tiến hành việc “đào” block này.

Cấu trúc một block được chia ra thành hai phần chính là *tiêu đề (header)* và *thân (body)*. Block header sẽ chứa các thông tin sau: phiên bản block, giá trị băm của block liền trước, giá trị băm của toàn bộ giao dịch trong block, dấu thời gian (timestamp), độ khó, giá trị nonce, và số lượng giao dịch. Block body là nơi chứa các giao dịch.

Việc đào block là một công việc cần rất nhiều sức mạnh tính toán để thực hiện. Bitcoin giải quyết vấn đề trùng chi bằng một máy chủ gắn nhãn thời gian và khái niệm *chứng cứ xử lý (proof-of-work)*. “Đào” ở đây thực sự không phải là chúng ta phải cầm một chiếc cuốc và “đào” một block. Đào block là một quá trình tăng giá trị nonce của block đó cho tới khi áp dụng hàm băm mật mã lên block header thì giá trị

băm của block sẽ được bắt đầu bởi một số lượng bit 0 nhất định. Sau khi tìm ra được con số này, nội dung các giao dịch bên trong block không thể nào thay đổi mà không phải đào lại block. Bên cạnh đó, bởi vì giá trị băm của block này cũng là một phần giá trị của block liền sau, ta cũng phải đào lại toàn bộ các block phía sau block đó. Sự liên kết các block với nhau như thế này tạo thành một chuỗi các block nối dài, hay còn được gọi bởi một cái tên phổ biến hơn là chuỗi block (*blockchain*).



Hình 2.2: Sự liên kết giữa các block với nhau [2]

Một điều cần được nhắc tới là số lượng bit 0 yêu cầu của giá trị băm block là không cố định mà dựa vào một giá trị được xác định dựa vào độ lớn của mạng lưới người đào bitcoin, giá trị này được gọi là *độ khó* (*difficulty*). Độ khó càng tăng cao, số lượng bit 0 yêu cầu càng nhiều, ngược lại, độ khó càng thấp, số lượng bit 0 yêu cầu càng ít đi. Tính toán được về việc phần cứng máy tính ngày càng mạnh mẽ và mạng lưới người đào bitcoin ngày càng lớn mạnh, hệ thống bitcoin được thiết kế để thay đổi độ khó một cách tự động sao cho chỉ có một block được đào thành công trong khoảng thời gian trung bình là 10 phút.

Khi một nút trong mạng đã đào thành công một block, nút đó sẽ tiến hành phát block vừa đào thành công cho các nút còn lại trong hệ thống. Việc đầu tiên mà một nút sẽ làm khi nhận được một block vừa được đào là sẽ đi kiểm chứng rằng toàn bộ giao dịch trong block là hợp lệ và không trùng chi, đồng thời, nút cũng sẽ tiến hành kiểm tra tính hợp lệ của block bằng cách thực hiện hàm băm cho block và kiểm tra số lượng bit 0 có đúng với yêu cầu hay không. Một đặc điểm của hệ thống blockchain là các nút sẽ luôn chọn chuỗi block dài nhất để tiếp tục thực hiện việc đào.

Bên cạnh việc đào block để ghi nhận các giao dịch vào blockchain, việc đào thành công một block cũng sẽ có một “phần thưởng” dành cho người đào thành công. Trước khi đào, người đào sẽ được quyền thêm một giao dịch đặc biệt vào block, giao dịch đó là một giao dịch gửi một lượng bitcoin tới trực tiếp ví bitcoin của người đào. Lượng bitcoin này được xem là phần thưởng dành cho họ khi đào block.. Bên cạnh giao dịch đặc biệt này, một phần thưởng khác dành cho người đào sẽ là tổng phí của các giao dịch trong block đó.

2.4. Tính an toàn và chống gian lận

2.4.1. Sự an toàn của một giao dịch bitcoin

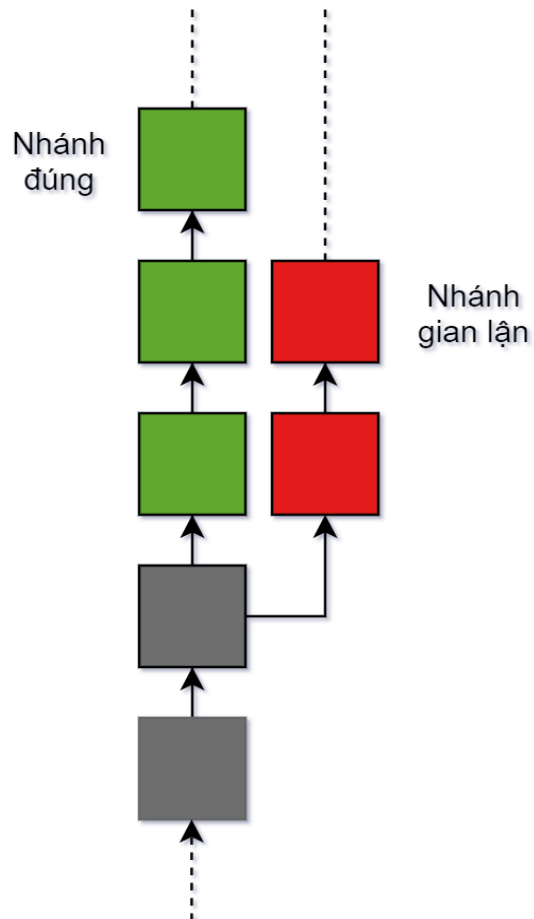
Với việc mỗi người dùng bitcoin đều nắm giữ cho mình một cặp khóa là khóa công khai và khóa bí mật, mỗi giao dịch đều yêu cầu người gửi tiền dùng khóa bí mật của mình ký lên giao dịch đó. Sau khi ký xong, chữ ký điện tử này sẽ được đính kèm vào thông tin của giao dịch và bởi vì khóa bí mật chỉ có một mình người gửi nắm giữ, cho nên nếu chúng ta thực hiện việc xác thực giao dịch sử dụng khóa công khai của người gửi thành công thì giao dịch được xem là hợp lệ.

2.4.2. Khả năng chống gian lận của blockchain

Một trường hợp có thể xảy ra trên blockchain là khi một kẻ gian tự tạo ra một block mới và tự đào block đó. Sau khi đào thành công, kẻ gian sẽ phát block này ra cho các nút đang đào trên mạng. Tuy nhiên, bởi vì mạng bitcoin là rất lớn nên sẽ xảy ra trường hợp một số nút sẽ nhận node gian lận trước, còn số nút còn lại sẽ nhận node đúng trước.

Vấn đề hiện tại sẽ là một cuộc đua tìm chứng cứ xử lý cho block tiếp theo. Và bởi vì số lượng nút xử lý nhánh đúng là rất nhiều, cho nên cuộc chạy đua này gần như là rất khó cho người gian lận.

Khả năng tạo một nhánh mới dài hơn nhanh đúng là gần một việc rất khó khăn và gần như không thể trừ khi nhánh gian lận có tổng lượng công suất đào là lớn so với nhánh đúng.



Hình 2.3: Hình minh họa cho việc “chạy đua” giữa hai nhánh blockchain

Chương 3

Bitcoin Core và libbitcoin

Cùng lúc công bố sách trắng của bitcoin, Satoshi Nakamoto còn công bố thêm một ứng dụng vận hành hệ thống, hay nút đầy đủ, mẫu mã nguồn mở có tên là Bitcoin, hay còn được biết tới với tên gọi là *Satoshi Client*, sau đó được đổi tên thành Bitcoin Core. Chương này sẽ đi qua về cách cài đặt và vận hành của Bitcoin Core. Bên cạnh đó, chương cũng sẽ đi qua về một hệ thống thư viện libbitcoin đồ sộ giúp phát triển bitcoin.

3.1. Giới thiệu về Bitcoin Core

Vào năm 2009, bên cạnh việc công bố sách trắng về bitcoin, Satoshi Nakamoto còn công bố thêm một mẫu cài đặt cho hệ thống bitcoin thông qua ứng dụng mã nguồn mở với tên đơn giản chỉ là Bitcoin. Kể từ phiên bản 0.9, Bitcoin được đổi tên lại thành Bitcoin Core [5]. Bitcoin Core được xây dựng dựa với mục đích là tạo ra một phiên bản mẫu của một nút đầy đủ, và cách vận hành chuẩn mực cho các nút bitcoin được sinh ra sau này bởi các nhà phát triển khác.

Bởi vì bitcoin được bảo trì và phát triển bởi cộng đồng nên ngày càng có nhiều *đề xuất cải tiến bitcoin* (Bitcoin Improvement Proposal – BIP) được đề ra². Các BIP này được công khai, giúp cho các nút đầy đủ khác ngoài Bitcoin Core có thể theo dõi và cài đặt các tính năng tương ứng.

Hiện nay, bên cạnh Bitcoin Core, các phiên bản nút bitcoin đầy đủ khác có thể kể đến như libbitcoin [6] và Armory [7]. Một nút được xem là đầy đủ nếu nó thực hiện việc xác minh toàn bộ giao dịch trên toàn bộ blockchain [8]. Chúng ta sẽ đi tìm hiểu về cách cài đặt và sử dụng Bitcoin Core ở phần tiếp theo.

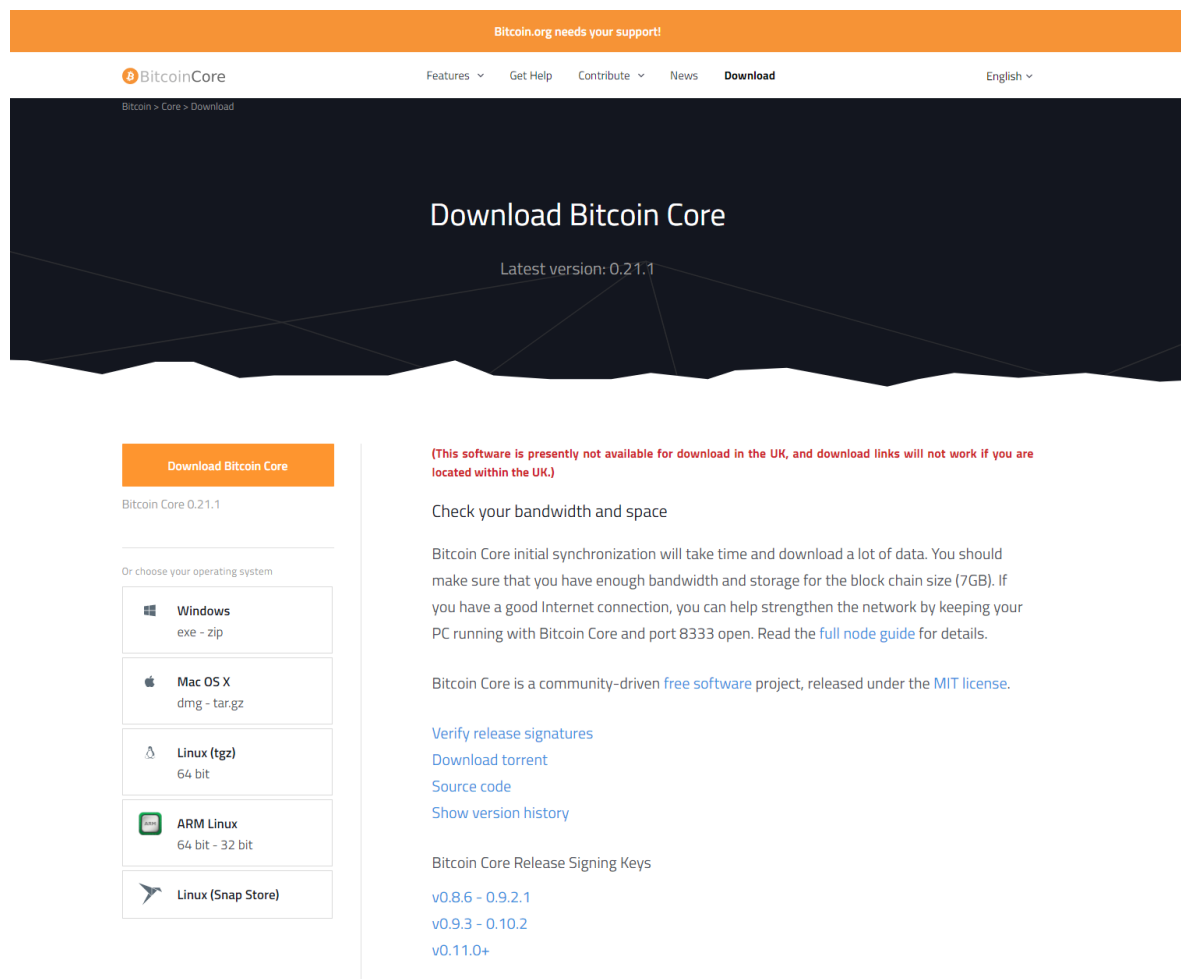
² Danh sách toàn bộ các BIP có thể được tìm thấy tại đây: <https://en.bitcoin.it/wiki/Category:BIP>.

3.2. Cài đặt Bitcoin Core

Bitcoin Core được cung cấp một trình cài đặt đơn giản do cộng đồng phát triển đóng gói và phân phối. Bên cạnh sử dụng phiên bản được phân phối, vì đây là chương trình mã nguồn mở nên ai cũng có thể đọc mã nguồn, tự biên dịch, và cài đặt.

3.2.1. Bản cài đặt do nhà phát triển đóng gói

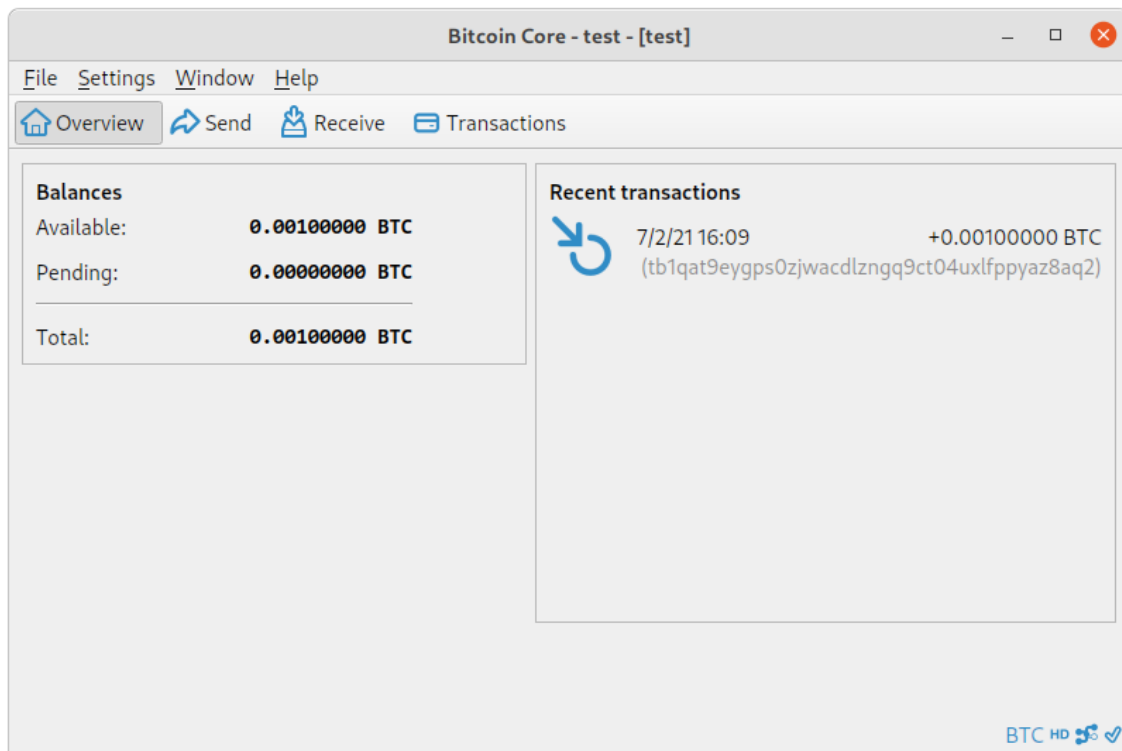
Đường dẫn tải về Bitcoin Core có thể được tìm thấy ở trang web sau: <https://bitcoin.org/en/download>. Tùy vào hệ điều hành mà người dùng đang sử dụng, ta sẽ tải về phiên bản tương ứng.



Hình 3.1: Trang tải về tập tin cài đặt Bitcoin Core

(Nguồn: <https://bitcoin.org/en/download>)

Sau khi cài đặt xong, người dùng có thể mở ứng dụng Bitcoin Core lên và sử dụng.



Hình 3.2: Giao diện ứng dụng Bitcoin Core

3.2.2. Tự biên dịch và cài đặt

Với một số người muốn tự biên dịch lại từ mã nguồn, vì Bitcoin Core là một ứng dụng mã nguồn mở nên việc tự cài đặt là hết sức dễ dàng. Hướng dẫn cài đặt dưới đây được áp dụng với hệ điều hành Ubuntu phiên bản 20.04.

Quá trình cài đặt có thể tóm gọn lại trong bốn bước sau:

- Cài đặt các thư viện cần thiết;
- Tải mã nguồn chương trình;
- Biên dịch chương trình;
- Cài đặt chương trình.

3.2.2.1. Cài đặt thư viện cần thiết

Trước khi tiến hành biên dịch Bitcoin Core, chúng ta cần phải cài đặt các thư viện tiên quyết, và xác định các tính năng mà mình muốn phiên bản mình biên dịch sử dụng, nhằm cài đặt thêm các thư viện bổ trợ.

Bảng dưới đây tóm tắt toàn bộ các thư viện mà Bitcoin Core sử dụng.

Bảng 3.1: Bảng tóm tắt các thư viện hỗ trợ quá trình biên dịch Bitcoin Core

Tên thư viện	Bắt buộc	Tính năng cung cấp
libboost	Có	Thư viện dùng để hỗ trợ cài đặt Bitcoin Core
libevent	Có	Thư viện dùng để xử lý mạng bất đồng bộ
miniupnpc	Không	Hỗ trợ nhảy tường lửa
libdb	Không	Hỗ trợ ví, chỉ cần cài nếu có biên dịch ví
qt	Không	Hỗ trợ cài đặt giao diện người dùng
libqrencode	Không	Hỗ trợ xuất kết quả bằng mã QR
univalue	Không	Hỗ trợ đọc và ghi JSON
libzmq3	Không	Hỗ trợ tạo thông báo
sqlite3	Không	Công cụ lưu trữ ví, chỉ cần cài nếu có biên dịch ví

Với hệ điều hành Ubuntu 20.04, ta có thể cài đặt những thư viện trên với những lệnh tương ứng sau:

- Cài đặt những công cụ phục vụ quá trình biên dịch:

```
sudo apt install build-essential libtool autotools-dev
automake pkg-config bsdmaintils python3
```

- Cài đặt thư viện bắt buộc:

```
sudo apt install libevent-dev libboost-system-dev
libboost-filesystem-dev libboost-test-dev libboost-
thread-dev
```

- Cài đặt các thư viện hỗ trợ có thể thực hiện bằng lệnh sau:

```
sudo apt install libsqlite3-dev libminiupnpc-dev
libzmq3-dev libqt5gui5 libqt5core5a libqt5dbus5
qttools5-dev qttools5-dev-tools libqrencode-dev libdb-
dev libdb++-dev
```

3.2.2.2. Tải mã nguồn Bitcoin Core

Mã nguồn của Bitcoin Core phiên bản mới nhất có thể tải ở đường dẫn sau: <https://github.com/bitcoin/bitcoin/releases>. Sau khi tải mã nguồn, chúng ta có thể giải nén mã nguồn và có thể tiến hành thực hiện việc biên dịch và cài đặt chương trình.

3.2.2.3. Biên dịch Bitcoin Core

Biên dịch Bitcoin Core cần phải thông qua một số bước nhất định nhằm cấu hình biên dịch trước khi chúng ta thực sự tiến tới việc biên dịch.

Cấu hình biên dịch

Cấu hình biên dịch có thể hiểu là quá trình chúng ta chọn những tùy chọn nào sẽ được biên dịch chung với Bitcoin Core. Một số tính năng hỗ trợ có thể kể đến như: hỗ trợ ví, hỗ trợ xuất mã QR, hỗ trợ giao diện đồ họa, ...

Một điểm cần lưu ý trong quá trình cài đặt Bitcoin Core đó là về Berkeley DB (thư viện libdb và libdb++). Nhằm giúp Bitcoin Core có tính tương thích ngược, các nhà phát triển Bitcoin Core khi cung cấp bản biên dịch sẵn sẽ biên dịch với phiên bản Berkeley DB 4.8. Tuy nhiên, trong quá trình cài thư viện, thì phiên bản Berkeley DB được cài sẽ là từ phiên bản 5.0 trở lên, dẫn tới việc tương thích ngược bị phá vỡ. Nếu một người muốn sử dụng Berkeley DB 4.8 thì họ phải tự biên dịch và cài đặt lại Berkeley DB 4.8 từ mã nguồn. Mặc định, quá trình cấu hình biên dịch chỉ kiểm tra và cho phép phiên bản 4.8 của Berkeley DB được sử dụng. Để sử dụng phiên bản mới hơn, chúng ta cần phải cấu hình lại quá trình biên dịch một chút.

Sau khi giải nén mã nguồn, một người có thể mở dòng lệnh và tiến hành cấu hình bằng các lệnh sau:

```
./autogen.sh
./configure --with-incompatible-bdb
```

Thấy rằng quá trình cấu hình bao gồm hai lệnh rời rạc nhau. Đầu tiên ta cần chạy lệnh `./autogen.sh` để tạo ra những tập tin cấu hình biên dịch. Sau đó, ta chạy lệnh `./configure` để tiến hành quá trình cấu hình. Thấy rằng ta có truyền thêm một tùy chọn là `--with-incompatible-bdb`, tùy chọn này là tùy chọn giúp chúng ta cấu

hình biên dịch với phiên bản Berkeley DB mới hơn 4.8. Để biết thêm các tùy chọn được hỗ trợ, ta chỉ cần chạy lệnh sau: `./configure --help`.

Biên dịch

Sau khi cấu hình xong, chúng ta đã có thể biên dịch được mã nguồn của Bitcoin Core. Để biên dịch mã nguồn, ta chỉ đơn giản chạy lệnh sau:

```
make
```

Tuy Bitcoin Core không phải là một chương trình lớn, nhưng thời gian biên dịch là thực sự không quá nhanh. Với máy tính hiện đại, chúng ta có thể tận dụng tối đa vi xử lý (CPU) đa nhân bằng cách thêm tùy chọn `-jN` để thực hiện việc biên dịch với N tiến trình biên dịch song song. Ví dụ ta muốn biên dịch song song 4 tiến trình, ta sẽ sửa lệnh bên trên lại thành:

```
make -j4
```

Giảm lượng bộ nhớ sử dụng trong quá trình biên dịch

Quá trình biên dịch là một quá trình cần rất nhiều tài nguyên của thiết bị. Bên cạnh sức mạnh xử lý của CPU, một loại tài nguyên khác không kém phần quan trọng là dung lượng bộ nhớ truy xuất ngẫu nhiên (RAM). Mặc định, khi chúng ta biên dịch, trình biên dịch mặc định sẽ là bộ biên dịch GNU GCC. Theo như thử nghiệm thực tế, một tiến trình biên dịch của GNU GCC có thể sử dụng lên tới 1.5 gigabyte dung lượng bộ nhớ RAM, và có thể dẫn tới tình trạng treo máy, không thể biên dịch được với một số máy tính có dung lượng RAM không đủ để cung cấp cho trình biên dịch.

Có hai cách để khắc phục vấn đề này [9]:

- *Cách 1:* Tinh chỉnh lại cấu hình biên dịch để sử dụng bộ nhớ ít hơn:

```
./configure CXXFLAGS="--param ggc-min-expand=1 --param ggc-minheapspace=32768"
```

- *Cách 2:* Sử dụng Clang thay cho GNU GCC:

```
./configure CXX=clang++ CC=clang
```

Với cách làm này, chúng ta cần phải cài thêm bộ biên dịch Clang:

```
sudo apt install clang
```


3.2.2.4. Cài đặt Bitcoin Core

Sau khi quá trình biên dịch hoàn tất, chúng ta có thể dễ dàng cài đặt Bitcoin Core chỉ bằng hai lệnh duy nhất:

```
sudo make install  
sudo ldconfig
```

3.3. Sử dụng Bitcoin Core

3.3.1. Cấu hình Bitcoin Core

Sau khi cài đặt thành công Bitcoin Core, chúng ta nên cấu hình phần mềm trước khi chạy. Bởi vì Bitcoin Core là một nút đầy đủ, chương trình sẽ tải toàn bộ blockchain về trên thiết bị lưu trữ cục bộ. Mặc định, nếu chúng ta không cấu hình mà khởi động Bitcoin Core ngay thì chương trình sẽ tải blockchain của mainnet, mạng bitcoin chính thức. Tại thời điểm viết báo cáo, blockchain mainnet đã vượt ngưỡng 350 gigabyte về mặt dung lượng [10], vì thế, nếu một người không có nhu cầu sử dụng mainnet thì việc khởi động ngay sẽ ảnh hưởng khá lớn tới dung lượng bộ nhớ cục bộ của họ.

Cấu hình đầu tiên chúng ta sẽ làm quen là cấu hình về mạng nào chúng ta sẽ sử dụng. Như đã trình bày ở trên, mặc định Bitcoin Core sẽ sử dụng mạng mainnet để hoạt động. Bên cạnh mainnet, chúng ta còn hai loại mạng nữa có thể sử dụng, đó là: *testnet* và *regtest*. Trong đó, testnet là mạng thử nghiệm dành hướng tới những nhà phát triển dự án trên blockchain có thể thử nghiệm ứng dụng của họ một cách an toàn, toàn bộ tính chất của testnet là giống hệt mainnet, chỉ khác một chỗ là đồng tiền trên testnet hoàn toàn không có giá trị. Dạng mạng thứ ba là regtest, đây là loại mạng cục bộ, chỉ tồn tại trong trên máy tính người sử dụng. Loại mạng này cũng được thiết kế hướng tới các nhà phát triển, bởi vì đây là mạng cục bộ nên họ có toàn quyền quyết định về việc sinh tiền, và tự đào block do chính mình tạo ra.

Để thay đổi cấu hình mặc định cho Bitcoin Core, chúng ta cần phải thay đổi nội dung tập tin *bitcoin.conf* nằm trong thư mục *.bitcoin* ở thư mục cá nhân của người

dùng (*home directory*). Để sử dụng testnet, chúng ta sẽ thêm một dòng `testnet=1` vào thư mục. Để sử dụng regtest, chúng ta thay dòng cấu hình trên thành `regtest=1`.

Một cấu hình khác hỗ trợ cho đề tài này sẽ là cấu hình tạo chỉ mục cho giao dịch. Mục đích của cấu hình này là để Bitcoin Core tạo chỉ mục cho các giao dịch trong quá trình tải blockchain về. Nếu có ta bật cấu hình này, sau này nếu ta có nhu cầu truy vấn thông tin giao dịch trong quá khứ thì Bitcoin Core vẫn có khả năng truy vấn được, nếu không bật, chúng ta chỉ truy vấn được những giao dịch liên quan tới ví do Bitcoin Core quản lý. Để bật cấu hình này, ta chỉ cần thêm dòng `txindex=1` vào tập tin cấu hình.

Nếu bộ nhớ lưu trữ của người dùng có phân chia nhiều phân vùng và họ muốn thay đổi thư mục lưu trữ blockchain cục bộ, họ có quyền thay đổi nó bằng tùy chọn `datadir=<path>` với *path* là đường dẫn tới thư mục mà người dùng muốn blockchain tải về được định vị ở đâu trên hệ thống.

Ngoài ba cấu hình đơn giản được trình bày ở trên, Bitcoin Core còn có rất nhiều cấu hình khác, để biết thêm thông tin về toàn bộ các cấu hình thì người dùng có thể chạy lệnh `bitcoind --help`.

3.3.2. Lần đầu khởi động Bitcoin Core

Như đã trình bày bên trên, mỗi lần Bitcoin Core khởi động, chương trình sẽ kiểm tra xem blockchain được lưu trữ cục bộ đã được cập nhật tới block mới nhất của mạng tương ứng hay chưa. Nếu chưa thì Bitcoin Core sẽ thực hiện tải những block còn thiếu. Sau khi quá trình kiểm tra và cập nhật hoàn tất, Bitcoin Core lúc này sẽ hoạt động như một nút đầy đủ, thực hiện việc lắng nghe các khối mới và thực hiện nhiệm vụ của một nút đầy đủ là xác minh giao dịch nằm trong block mới được nhận.

3.3.3. Giao tiếp với Bitcoin Core

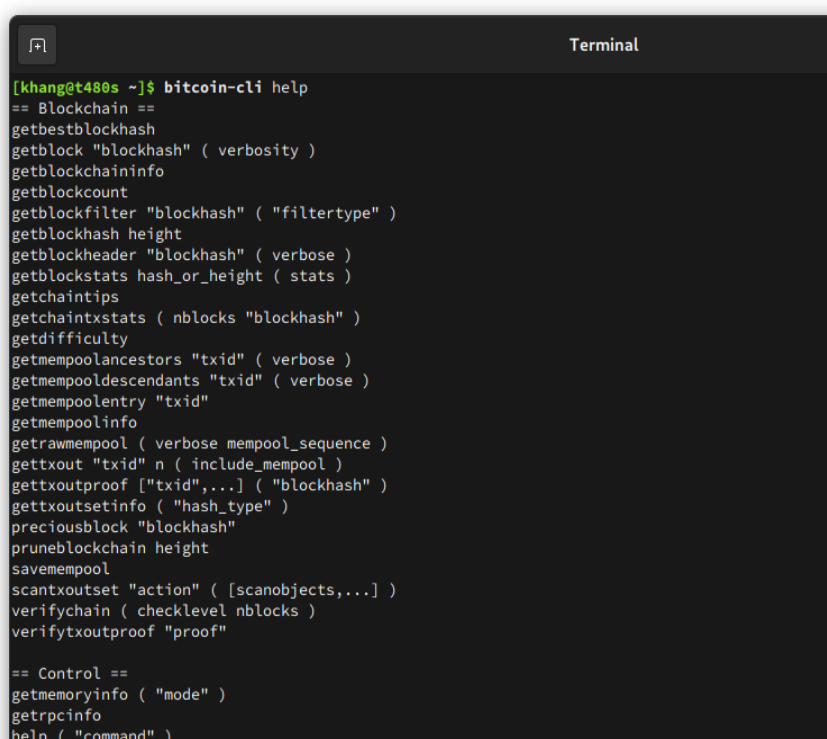
Sau khi Bitcoin Core khởi động thành công, chương trình lúc này sẽ mở cổng 8333 hoặc 18333 tương ứng cho mainnet hoặc testnet để lắng nghe truy vấn tới chương trình để thực hiện truy vấn và trả về kết quả. Giao thức của công việc này là

giao thức JSON-RPC³. Bằng việc sử dụng giao thức JSON-RPC, chỉ cần ứng dụng có tồn tại một JSON-RPC client để giao tiếp với giao thức JSON-RPC nằm trong Bitcoin Core, Bitcoin Core có thể phục vụ ứng dụng với bộ chức năng phong phú mà Bitcoin Core hỗ trợ.

Để thử nghiệm giao tiếp này, một chương trình hỗ trợ được đính kèm theo trong lúc trong lúc mình cài Bitcoin Core là *bitcoin-cli*. Sử dụng *bitcoin-cli*, ta có thể cấu tạo một câu lệnh JSON-RPC tới trực tiếp Bitcoin Core mà không cần phải sử dụng ứng dụng bên thứ ba. Để biết được toàn bộ các giao thức mà Bitcoin Core hỗ trợ, ta chỉ cần thực hiện lệnh:

```
bitcoin-cli help
```

Đầu ra của lệnh này sẽ giống hình bên dưới.



```
[khang@t480s ~]$ bitcoin-cli help
== Blockchain ==
getbestblockhash
getblock "blockhash" ( verbosity )
getblockchaininfo
getblockcount
getblockfilter "blockhash" ( "filtertype" )
getblockhash height
getblockheader "blockhash" ( verbose )
getblockstats hash_or_height ( stats )
getchaintips
getchaintxstats ( nblocks "blockhash" )
getdifficulty
getmempoolancestors "txid" ( verbose )
getmempooldescendants "txid" ( verbose )
getmempoolentry "txid"
getmempoolinfo
getrawmempool ( verbose mempool_sequence )
gettxout "txid" n ( include_mempool )
gettxoutproof ["txid",...] ( "blockhash" )
gettxoutsetinfo ( "hash_type" )
preciousblock "blockhash"
pruneblockchain height
savemempool
scantxoutset "action" ( [scanobjects,...] )
verifychain ( checklevel nblocks )
verifytxoutproof "proof"

== Control ==
getmemoryinfo ( "mode" )
getrpcinfo
help ( "command" )
```

Hình 3.3: Xem những hàm Bitcoin Core hỗ trợ

³ JSON-RPC là một giao thức thực hiện lệnh từ xa (remote procedure call – RPC) mà lời gọi được quy định theo định dạng JSON.

Sau khi có đầu ra của lệnh `getrawtransaction`, ta có thể lấy nó và đưa vào hàm `decoderawtransaction` để thấy được nội dung cụ thể của giao dịch đó.

```

[~] Terminal
[khang@2480s ~]$ bitcoin-cli decoderawtransaction 0100000000130d73649da9ed5ba685a5d5b7c4014be168f90827c0104a9eda74728a7157c1000000
0b483045922100deca2ec2e58599d99bf86520d3c13efa3b4d80604248ca07e430e768dd4c524b022030e3a7dcfc80ba3652cecec336987f87f89a1d257b25e18
8d4477eff680a00121026a0c583ad692cd345815d898b4ed3de261be6826d249af40397f171fcb4c24cffffffffff02983a000000000000001976a9144538024caf2f
379dee784ccc488a53139488ac14a01000000000001976a914746ca5f027d7abb2351ef4501f85282c2ecf3c6a8ac00000000
{
  "txid": "938d60512bcd3d61411da4bd3c015ed27c596faa999cf47a2989938d314924",
  "hash": "938d60512bcd3d61411da4bd3c015ed27c596faa999cf47a2989938d314924",
  "version": 1,
  "size": 226,
  "vsize": 226,
  "weight": 904,
  "locktime": 0,
  "vin": [
    {
      "txid": "c157718a7274da964a10c02708f7668b14e01c4b7d5506a68bad68eda4036d730",
      "vout": 0,
      "scriptSig": {
        "asm": "3045022100deca2ec2e58599d99bf86520d3c13efa3b4d80604248ca07e430e768dd4c524b022030e3a7dcfc80ba3652cecec336987f87f89
d257b25e18f68d4477eff680a0[ALL] 026a0c583ad692cd345815d898b4ed3de261be6826d249af40397f171fcb4c24c",
        "hex": "483045022100deca2ec2e58599d99bf86520d3c13efa3b4d80604248ca07e430e768dd4c524b022030e3a7dcfc80ba3652cecec336987f87f89
a1d257b25e18f68d4477eff680a00121026a0c583ad692cd345815d898b4ed3de261be6826d249af40397f171fcb4c24c"
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.00015000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 4538024caf2fd1dec3b79dee704ccc488a531394 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9144538024caf2fd1dec3b79dee704ccc488a53139488ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "mnpYrZ7Y6DFioVBMRLTz4BQ6WXXUYECZ"
        ]
      }
    },
    {
      "value": 0.00084500,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 746ca5f027d7abb2351ef4501f85282c2ecf3c6a OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914746ca5f027d7abb2351ef4501f85282c2ecf3c6a88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "8rYqKFZnNHVaitPF24Sw4QTVeshxx7EPv"
        ]
      }
    }
  ]
}
[khang@2480s ~]$

```

Hình 3.6: Giải mã nội dung giao dịch

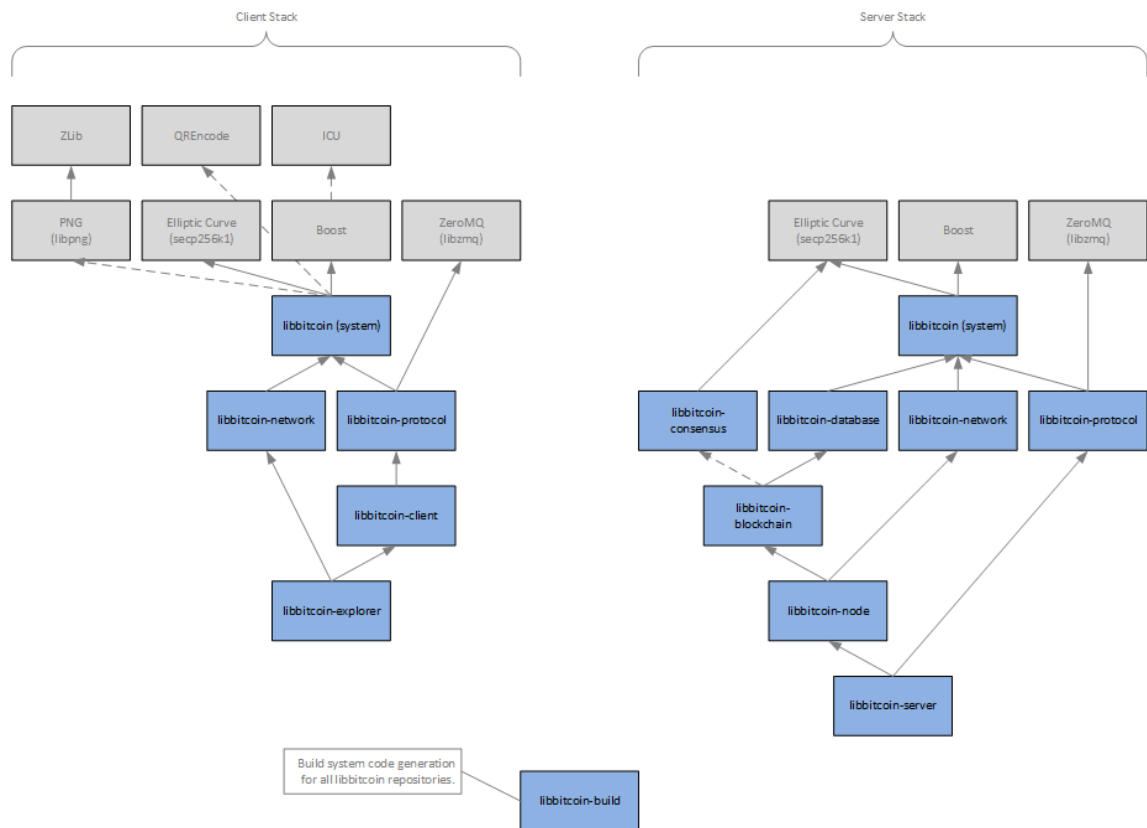
3.4. Hệ thống thư viện libbitcoin

3.4.1. Giới thiệu về libbitcoin

Libbitcoin vừa là một phiên bản nút đầy đủ, vừa là một hệ thống thư viện giúp lập trình bitcoin. Theo như giới thiệu trên trang chủ [6], libbitcoin là một hệ thống thư viện đa nền tảng hỗ trợ việc lập trình bitcoin bất đồng bộ.

Libbitcoin không chỉ là một thư viện riêng lẻ, libbitcoin là một hệ thống thư viện, bao gồm rất nhiều thư viện nhỏ hơn phục vụ từng mục đích khác nhau. Sơ đồ phụ thuộc và quan hệ các thư viện có thể hình như sơ đồ bên dưới

Libbitcoin Version 3



Hình 3.7: Sơ đồ quan hệ giữa các thư viện thuộc hệ thống libbitcoin

(Nguồn: <https://github.com/libbitcoin/libbitcoin-system/wiki>)

3.4.2. Cách cài đặt và sử dụng libbitcoin-system

Tuy hệ thống libbitcoin có rất nhiều thư viện, đề tài này chỉ dùng đến một thư viện duy nhất là thư viện *libbitcoin-system*. Theo như sơ đồ, thư viện này không phụ thuộc vào các thư viện thuộc hệ thống libbitcoin nào khác.

3.4.2.1. Cài đặt thư viện

Quá trình cài đặt libbitcoin-system tương tự như quá trình cài đặt của Bitcoin Core. Vì lý do này, quá trình cài đặt thư viện sẽ được trình bày một cách ngắn gọn.

Đầu tiên, chúng ta cần điểm qua một số thư viện cần thiết cho quá trình cài đặt libbitcoin-system.

Bảng 3.2: Bảng tóm tắt các thư viện hỗ trợ quá trình biên dịch libbitcoin-system

Tên thư viện	Bắt buộc	Tính năng cung cấp
secp256k1	Có	Hỗ trợ các thao tác liên quan đến thuật toán mã hóa đường cong elliptic
boost	Có	Thư viện libbitcoin sử dụng để cài đặt các tính năng liên quan tới mạng, cấu trúc dữ liệu, ...
zeromq	Có	Thư viện libbitcoin sử dụng để cài đặt hệ thống thông báo
icu	Không	Hỗ trợ bảng mã Unicode (dùng chủ yếu cho BIP-038, BIP-039 và ví Electrum [11])
qrencode	Không	Hỗ trợ mã hóa QRCode (chỉ sử dụng trong libbitcoin-explorer)
png	Không	Hỗ trợ xuất PNG (dùng cho xuất hình mã QR)

Để cài đặt toàn bộ các công cụ biên dịch và thư viện hỗ trợ, ta cần chạy lệnh sau:

```
sudo apt-get install build-essential autoconf automake libtool  
pkg-config libsecp256k1-dev libboost-all-dev libzmq3-dev icu-  
devtools libqrencode-dev libpng-dev
```

Sau khi cài đặt xong mọi thứ cần thiết, chúng ta sẽ tiến hành tải mã nguồn của thư viện, sau đó là cấu hình biên dịch và tiến hành biên dịch ngay sau đó. Mã nguồn của thư viện có thể được tải ở trang web sau: <https://github.com/libbitcoin/libbitcoin-system/releases>.

Sau khi tải mã nguồn, chúng ta tiến hành giải nén, sau đó mở ứng dụng dòng lệnh và thay đổi thư mục hiện hành thành thư mục chứa mã nguồn. Quá trình biên dịch với toàn bộ các tính năng mà thư viện hỗ trợ được diễn ra lần lượt theo thứ tự sau:

```
./autogen.sh
./configure --with-icu --with-qrencode --with-png
make
sudo make install
sudo ldconfig
```

Sau khi cài đặt, để chương trình của chúng ta có thể sử dụng libbitcoin được, ta cần phải thêm đường dẫn biến môi trường cho pkg-config để có thể nhận diện được libbitcoin bằng lệnh sau:

```
export
PKG_CONFIG_PATH="${PKG_CONFIG_PATH}:/usr/local/lib/pkgconfig"
```

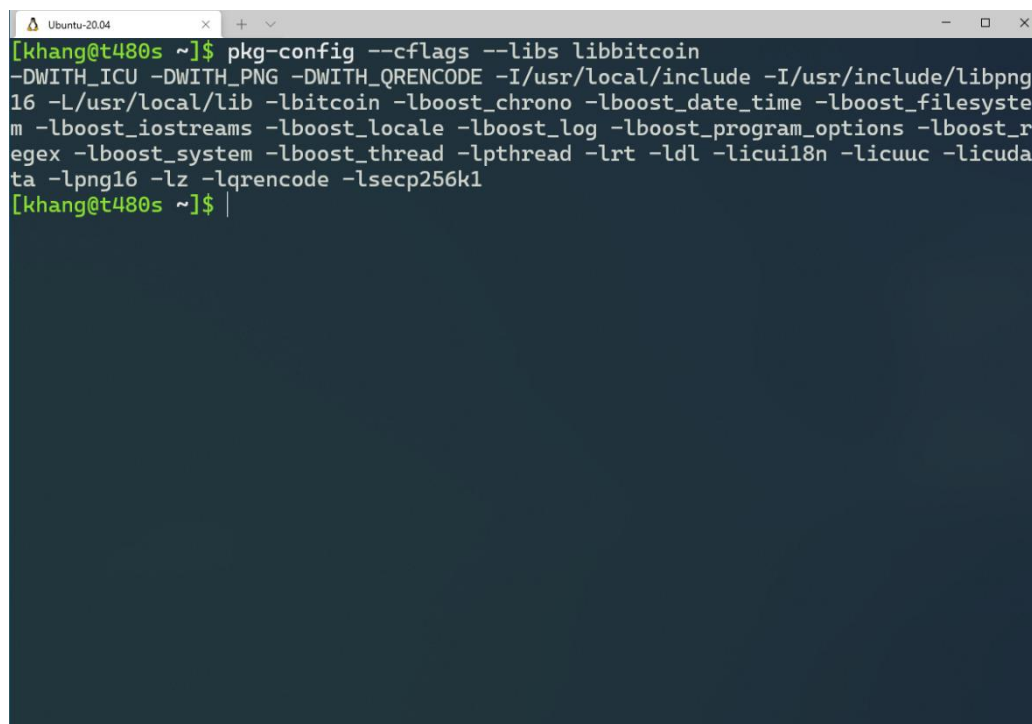
Tuy nhiên nếu chỉ chạy lệnh này thì sau khi khởi động lại máy sẽ bị mất, để giữ cho lệnh chạy qua mỗi lần khởi động lại máy, ta cần chạy thêm lệnh

```
echo export
PKG_CONFIG_PATH="${PKG_CONFIG_PATH}:/usr/local/lib/pkgconfig" >>
~/.profile
```

Sau khi thực hiện xong quá trình cài đặt thư viện, ta có thể xác minh rằng thư viện đã được cài đặt một cách đầy đủ bằng cách sử dụng câu lệnh sau:

```
pkg-config --libs --cflags libbitcoin
```


Nếu lệnh xuất ra màn hình tương tự với hình dưới thì tức là mình đã cài đặt libbitcoin-system thành công.



```
[khang@t480s ~]$ pkg-config --cflags --libs libbitcoin
-DWITH_ICU -DWITH_PNG -DWITH_QRENCODE -I/usr/local/include -I/usr/include/libpng16 -L/usr/local/lib -lbitcoin -lboost_chrono -lboost_date_time -lboost_filesystem -lboost_iostreams -lboost_locale -lboost_log -lboost_program_options -lboost_regex -lboost_system -lboost_thread -lpthread -lrt -ldl -licui18n -licuuc -licudata -lpng16 -lz -lqrencode -lsecp256k1
[khang@t480s ~]$
```

Hình 3.8: Nội dung được xuất ra khi cài đặt thành công libbitcoin-system

3.4.2.2. Sử dụng thư viện

Để sử dụng thư viện, lập trình viên chỉ cần khai báo thư viện như sau:

```
1 #include <bitcoin/bitcoin.hpp>
```

Sau khi đã cài đặt xong chương trình, khi biên dịch, ta cần phải thêm một số tùy chọn để trình biên dịch có thể tìm thấy thư viện như sau:

```
g++ source.cpp $(pkg-config --cflags --libs libbitcoin)
```

Chương 4

Khóa và địa chỉ

Địa chỉ có thể hình dung như là tài khoản ngân hàng, còn *khóa* thì có thể hình dung như là mã PIN của thẻ ATM. Khóa và địa chỉ là hai khái niệm dính liền nhau và không kém phần quan trọng trong hệ thống bitcoin. Chương 4 sẽ đi vào tìm hiểu nền tảng mật mã, một trong những lĩnh vực hóc búa nhưng không kém phần thú vị, đã giúp tạo nên hai khái niệm này. Sau đó, chương sẽ đi vào trình bày cách thức tạo khóa và địa chỉ, cũng như một số định dạng khóa và địa chỉ phổ biến. Cuối cùng, chương cũng có cung cấp mã nguồn mẫu để mình có thể tự tạo khóa và mật mã.

4.1. Nền tảng mật mã

Việc ta sở hữu lượng tài sản bitcoin được xác định thông qua *khóa điện tử*, *địa chỉ bitcoin*, và *chữ ký điện tử* [12].

Khóa điện tử sẽ không được lưu trên một trung tâm dữ liệu, mà thay vào đó sẽ được lưu trữ trong một chương trình độc lập ở phía người dùng, hay còn được gọi là một ví. Các khóa này có thể được sinh ra một cách độc lập, không liên quan tới blockchain, và không cần chúng ta phải có kết nối tới mạng internet để tạo khóa. Vì lý do này, việc nghe lén quá trình tạo khóa là gần như không thể, trừ khi thiết bị của người dùng đã bị tổn hại hoặc kẻ gian có quyền truy cập vật lý. Khóa điện tử giúp chúng ta giải quyết một số bài toán trong bitcoin như sự tin tưởng trong mạng lưới phân tán, chứng minh tính sở hữu, và thực hiện các mô bảo mật bằng mật mã [12].

Khóa điện tử này được biểu diễn bằng một cặp khóa bao gồm khóa bí mật và khóa công khai. Khóa bí mật là thành phần giúp chúng ta mở khóa tài sản, chứng minh sự sở hữu và là người thực hiện giao dịch. Trong khi đó, địa chỉ được tạo thành từ khóa công khai.

4.1.1. Sơ lược mã hóa công khai

Vào năm 1976, ý tưởng mã hóa công khai lần đầu tiên được phát biểu bởi ba nhà khoa học Martin Hellman, Ralph Merkle, và Whitfield Diffie tại Đại học Stanford [13]. Trước đây, với những thuật toán mã hóa đối xứng, chỉ có thuật toán sử dụng để mã hóa được công khai trong khi khóa phải được giữ bí mật. Ý tưởng đằng sau của mã hóa công khai là một người không chỉ có thể công khai thuật toán mã hóa họ dùng, họ còn có thể công khai khóa của họ. Một người gửi tin muốn gửi một bản tin mã hóa an toàn tới người nhận có thể sử dụng *khóa công khai* mà người nhận cung cấp, áp dụng thuật toán mã hóa được người nhận công khai sử dụng. Sau khi người nhận nhận được bản tin đã được mã hóa, họ có thể áp dụng thuật toán giải mã sử dụng một khóa thứ hai, có tên là *khóa bí mật*, của họ. Với thuật toán này, ai cũng có thể gửi tin an toàn tới người nhận và chỉ duy nhất người nhận mới có thể giải mã được nó.

Tính đến hiện nay, có rất nhiều thuật toán mã hóa công khai được sinh ra, một số thuật toán mã hóa công khai có thể kể đến như: Thuật toán RSA, giao thức thỏa thuận khóa có xác thực YAK, hay mật mã đường cong elliptic, ...

Hai ứng dụng phổ biến của thuật toán mã hóa công khai có thể kể đến là: *mã hóa* và *chữ ký điện tử*.

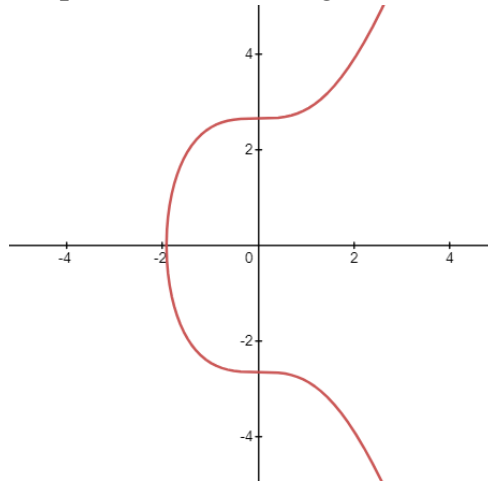
4.1.2. Thuật toán mã hóa đường cong elliptic

Bitcoin sinh khóa sử dụng thuật toán mã hóa đường cong elliptic. Phần này em sẽ trình bày sơ lược về những tính chất cơ bản thuật toán mã hóa đường cong elliptic.

Bitcoin sử dụng một phiên bản cụ thể của thuật toán mã hóa đường cong elliptic được biết đến là secp256k1 [14]. Theo tiêu chuẩn này, đường cong elliptic được sử dụng có công thức như sau:

$$y^2 = x^3 + 7 \tag{4.1}$$

Đường cong trên khi phác họa lên trường số thực sẽ có hình như sau:



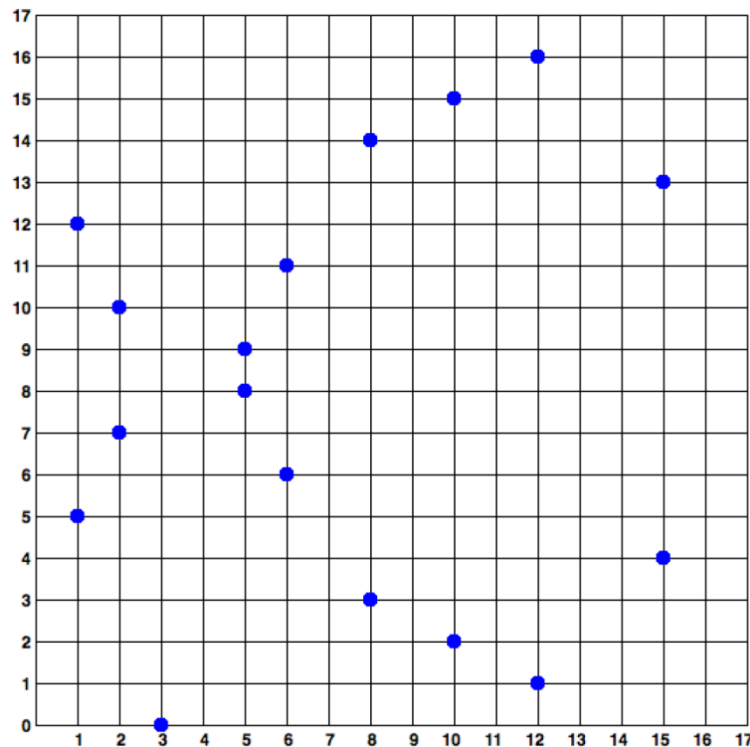
Hình 4.1: Hình ảnh đường cong elliptic trên trường số thực

(Nguồn: <https://www.desmos.com/calculator/ialhd71we3>)

Tuy nhiên, trên thực tế, vì đường cong elliptic này được biểu diễn trên tập hữu hạn \mathbb{F}_p nên việc phác họa là rất khó vì các điểm thuộc đường cong sẽ nhìn như các điểm rải rác ngẫu nhiên. Công thức đường cong từ đó được viết lại thành:

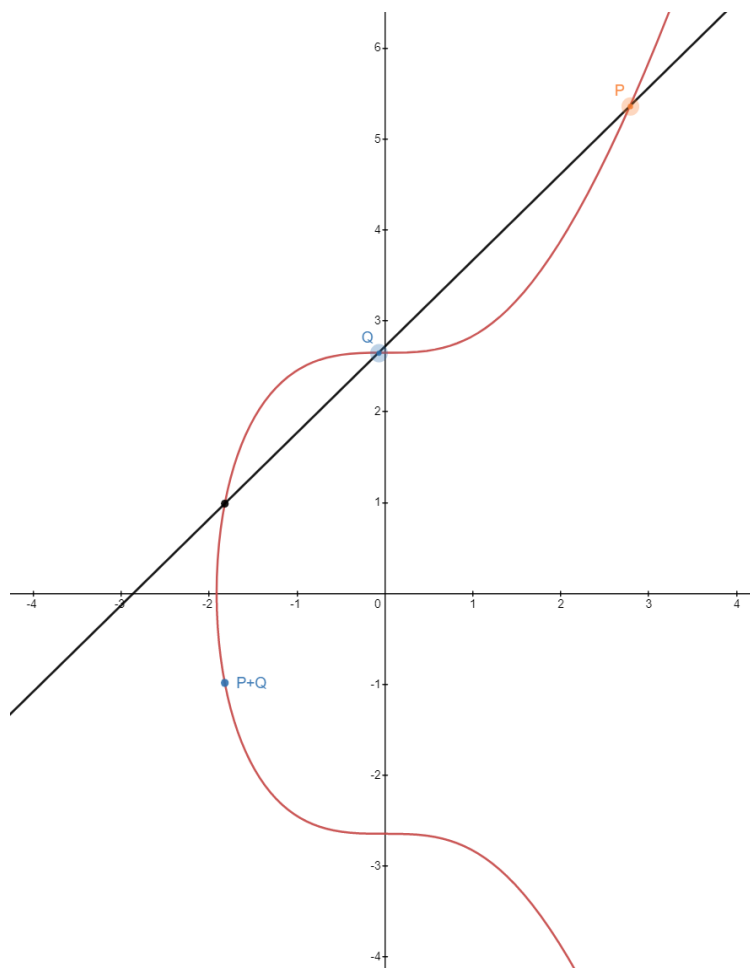
$$y^2 \bmod p = x^3 + 7 \bmod p$$

4.2



Hình 4.2: Hình ảnh đường cong elliptic với $p = 17$ [12]

Cụ thể, giá trị $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ là giá trị được sử dụng [14]. Trong đường cong elliptic, ta có thể lấy hai điểm $P(x_1, y_1)$ và $Q(x_2, y_2)$ thuộc đường cong và thực hiện phép cộng hai điểm lại như bình thường, điểm được sinh ra sẽ là một điểm thuộc đường cong elliptic. Trên đồ thị hàm số, điểm này có thể được xác định dễ dàng bằng cách kẻ một đường thẳng đi qua P và Q , đường thẳng này sau đó sẽ cắt đường cong tại một điểm (x_3, y_3) nào đó. Điểm ảnh của (x_3, y_3) qua trục Ox , tức điểm $(x_3, -y_3)$, sẽ là tổng của P và Q .



Hình 4.3: Hình minh họa tổng của hai điểm trên đường cong elliptic

(Nguồn: <https://www.desmos.com/calculator/ialhd71we3>)

Một số trường hợp đặc biệt có thể xảy ra như sau:

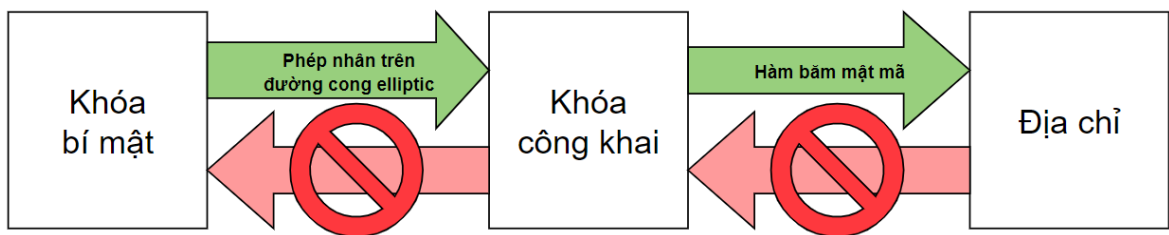
- Trong trường hợp P và Q trùng nhau: điểm $P + Q$ sẽ sử dụng cách tính khác nâng cao hơn;

- Trong trường hợp P và Q đối xứng qua trục Ox : lúc này $P + Q$ sẽ được gọi là *điểm vô cùng* (*point at infinity*). Điểm này mang ý nghĩa giống với điểm $(0, 0)$ trên hệ trục tọa độ Oxy bình thường.

Sau khi đã có phép cộng, phép nhân một số nguyên k cho một điểm thuộc đường cong elliptic chỉ là phép cộng điểm đó liên tiếp k lần.

4.2. Khóa

Như đã trình bày bên trên, bitcoin sử dụng hai loại khóa là *khóa bí mật* và *khóa công khai*. Mỗi quan hệ giữa khóa bí mật, khóa công khai, và địa chỉ có thể được biểu diễn như hình bên dưới.



Hình 4.4: Mối quan hệ giữa khóa bí mật, khóa công khai, và địa chỉ Bitcoin

Để ý rằng toàn bộ các mối quan hệ đều là một quan hệ một chiều mà không có chiều ngược lại. Vì lý do này, chúng ta có thể công khai địa chỉ bitcoin của chúng ta mà không cần phải lo lắng rằng việc đó đồng thời cũng tiết lộ khóa công khai, hay thậm chí là khóa bí mật của chúng ta.

4.2.1. Khóa bí mật

Khóa bí mật chỉ là một số nguyên ngẫu nhiên. Việc sở hữu khóa bí mật là minh chứng cho việc sở hữu tài sản bitcoin của người dùng. Khóa bí mật được dùng để tạo chữ ký điện tử, một phần bắt buộc khi muốn dùng bitcoin, nhằm chứng minh sự sở hữu của lượng bitcoin đó. Khóa bí mật phải luôn được giữ bí mật, nếu không, tình trạng mất mát tài sản là một điều hoàn toàn có thể xảy ra.

4.2.1.1. Tạo khóa bí mật

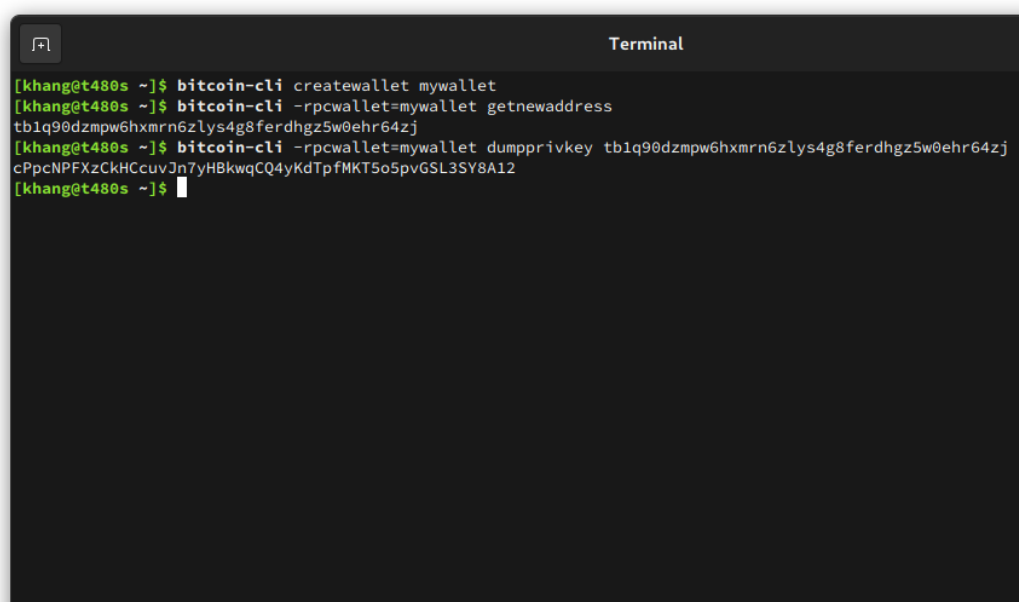
Tạo khóa bitcoin chỉ đơn giản là chọn ngẫu nhiên một số nguyên dương 256 bit. Một trong những điều kiện cần thiết để tạo khóa bí mật tốt đó là chúng ta phải tìm

được một trình sinh số ngẫu nhiên an toàn. Trình sinh số ngẫu nhiên được gọi là “an toàn” khi nó không thể dự đoán và không thể lặp lại.

4.2.1.2. Ví dụ về khóa bí mật sử dụng Bitcoin Core

Để tạo khóa sử dụng Bitcoin Core, đầu tiên chúng ta cần phải khởi tạo một ví mới do Bitcoin Core quản lý bằng câu lệnh `creatwallet`. Sau khi có ví, chúng ta có thể sử dụng lệnh `getnewaddress` để Bitcoin Core tạo một địa chỉ bitcoin mới. Địa chỉ bitcoin này sẽ được in ra màn hình dưới dạng mã hóa *Base58Check* có tên là *Wallet Import Format* (WIF). Định dạng *Base58Check* và WIF sẽ được trình bày ở phần 4.3.

Sau khi có được địa chỉ bitcoin, ta có thể sử dụng lệnh `dumpprivkey` để truy vấn khóa bí mật cho địa chỉ tương ứng.



```
Terminal
[khang@t480s ~]$ bitcoin-cli createwallet mywallet
[khang@t480s ~]$ bitcoin-cli -rpcwallet=mywallet getnewaddress
tb1q90dzmpw6hxmnrn6zlys4g8ferdhgz5w0ehr64zj
[khang@t480s ~]$ bitcoin-cli -rpcwallet=mywallet dumpprivkey tb1q90dzmpw6hxmnrn6zlys4g8ferdhgz5w0ehr64zj
cPpcNPFXzCkHCcuvJn7yHBkwqCQ4yKdTpFMKT5o5pvGSL3SY8A12
[khang@t480s ~]$
```

Hình 4.5: Tạo khóa bí mật sử dụng Bitcoin Core

4.2.2. Khóa công khai

Khóa công khai có thể được tính từ khóa bí mật thông qua phép nhân trên đường cong elliptic sử dụng phép nhân sau:

$$K = k \cdot G \quad 4.3$$

Trong đó, k là khóa bí mật, G là *điểm sinh*⁴, và K là khóa công khai. Bởi vì điểm G luôn cố định cho toàn bộ hệ thống bitcoin, một khóa bí mật k luôn luôn cho ra duy nhất một khóa công khai K . Bên cạnh đó, để tính được khóa bí mật k khi biết khóa công khai K là điều gần như không thể⁵.

4.3. Địa chỉ

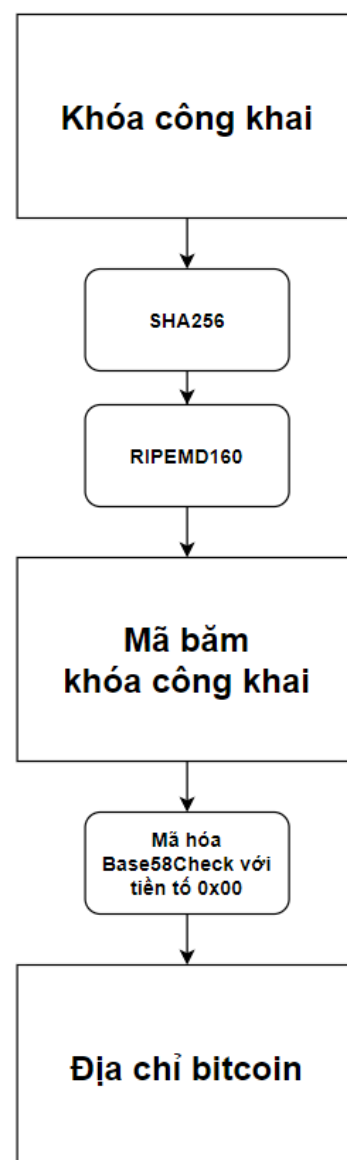
Một địa chỉ bitcoin chỉ đơn giản là một chuỗi các ký tự mà bạn sẽ chia sẻ cho ai muốn gửi bitcoin tới bạn. Địa chỉ bitcoin được tạo ra từ khóa công khai sau khi áp dụng hàm băm mật mã. Cụ thể, hàm băm được sử dụng để tạo ra địa chỉ bitcoin là một tổ hợp hai hàm băm SHA256 và RIPEMD160.

Từ khóa công khai K , chúng ta tính giá trị băm SHA256, sau đó ta tiếp tục tính giá trị băm RIPEMD160 của giá trị băm SHA256 vừa có được. Kết quả cuối cùng ta có là một giá trị băm có độ lớn là 160 bit. Quá trình băm có thể được biểu diễn như sau:

$$A = \text{RIPEMD160}(\text{SHA256}(K)) \quad 4.4$$

với K là khóa công khai, và A là mã băm của khóa công khai. Quy trình này còn có tên gọi khác là HASH160.

Giá trị băm này sau đó sẽ được mã hóa dưới dạng Base58Check⁶ để tạo ra địa chỉ hoàn chỉnh. Hình bên là sơ đồ cho quá trình tạo địa chỉ bitcoin từ khóa công khai.



Hình 4.6: Quá trình tạo địa chỉ bitcoin

⁴ Điểm sinh G là một điểm có tọa độ rất lớn, thỏa mãn đường cong elliptic. Giá trị của G có thể được tìm thấy tại: <https://en.bitcoin.it/wiki/Secp256k1>.

⁵ Theo như Andreas M. Antonopoulos viết [12], bài toán tìm khóa bí mật từ khóa công khai cho thuật toán mã hóa đường cong elliptic là bài toán *Lôgarit rời rạc* (Discrete logarithm problem), một bài toán cho đến ngày nay vẫn chưa có thuật toán giải quyết một cách hiệu quả.

⁶ Gần đây hơn, có một loại địa chỉ mới được đưa vào sử dụng là địa chỉ *bech32*, được đề xuất trong BIP-0173. Loại địa chỉ này hướng tới giải quyết một số vấn đề mà loại địa chỉ bitcoin ban đầu còn gặp phải. Nguồn: https://en.bitcoin.it/wiki/BIP_0173.

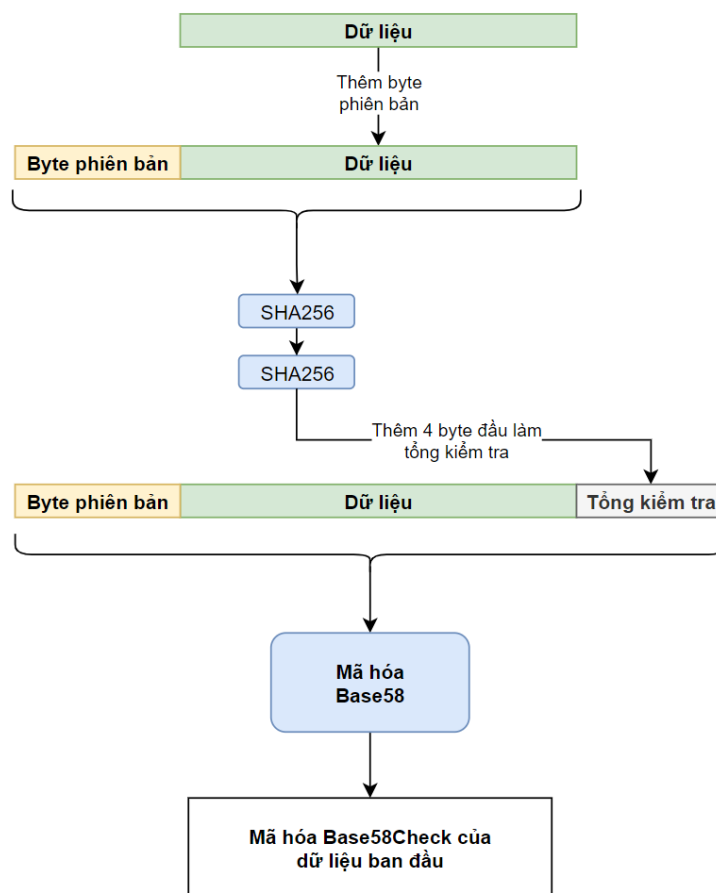
4.3.1. Định dạng mã hóa Base58 và Base58Check

Với nhu cầu thể hiện chuỗi nhị phân dưới dạng một chuỗi có thể đọc được, ta cần phải tìm đến các định dạng mã hóa. Trong bitcoin, nhiều thành phần sử dụng hai định dạng mã hóa là Base58 và Base58Check để mã hóa. Phần này sẽ đi tìm hiểu về cách thức hoạt động của hai định dạng mã hóa này.

Thông thường, định dạng Base64 được sử dụng khá phổ biến. Tuy nhiên, với bitcoin, những giá trị cần mã hóa đa phần là những thông tin quan trọng như địa chỉ và khóa. Vì thế, nếu sử dụng Base64 thì sẽ dễ bị nhầm lẫn một ký tự với nhau trong quá trình nhập dữ liệu. Base58 được sinh ra để giải quyết vấn đề này. Bảng ký tự của Base58 chỉ khác với bảng ký tự của Base64 một số điểm là không tồn tại các ký tự sau: số 0, ký tự O (o hoa), l (L thường), I (i hoa), và hai ký tự “+” và “/”.

Để thêm cơ chế phòng ngừa sai sót trong quá trình nhập liệu, Base58Check là một dạng mã hóa Base58 có đính kèm theo một đoạn mã kiểm tra. Trong bitcoin, quy trình mã hóa một đoạn dữ liệu dưới dạng Base58Check như sau:

- Đầu tiên, ta cần thêm vào đầu đoạn dữ liệu một byte phiên bản (*version byte*) hay còn được gọi là byte tiền tố;
- Tiếp theo, ta sẽ áp dụng thuật toán SHA256 hai lần toàn bộ đoạn dữ liệu sau khi thêm byte phiên bản. Sau đó, ta lấy 4 byte dữ liệu đầu trong giá trị băm đó và thêm vào sau đoạn dữ liệu;
- Cuối cùng, ta chỉ cần áp dụng mã hóa Base58, đầu ra sẽ là đoạn dữ liệu được mã hóa dưới định dạng Base58Check.



Hình 4.7: Quy trình mã hóa Base58Check

Trong bitcoin, đa số các dữ liệu được trả về cho người dùng sẽ thuộc định dạng Base58Check. Lý do một byte phiên bản cần được thêm vào trước đoạn dữ liệu là vì khi ta áp dụng mã hóa Base58, byte này sẽ cho ra một ký tự nhất định, dẫn tới việc xác định loại dữ liệu dễ dàng hơn.

Bảng 4.1: Một số loại dữ liệu áp dụng Base58

Loại dữ liệu	Byte tiền tố (hệ 16)	Ký tự đầu tiên của kết quả sau khi áp dụng Base58
Địa chỉ bitcoin	0x00	1
Địa chỉ bitcoin testnet	0x6F	m hoặc n
Khóa bí mật WIF (cả thường và nén)	0x80	5, K, hoặc L

4.3.2. Định dạng khóa

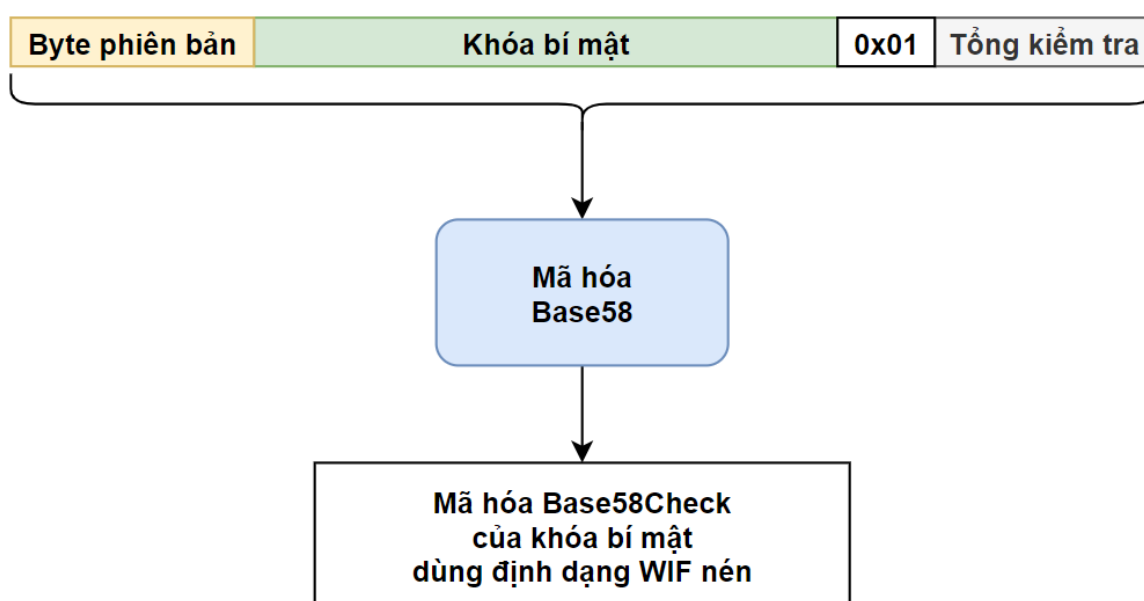
Khóa bí mật và khóa công khai của bitcoin đều có thể biểu diễn dưới một số định dạng nhất định. Chúng ta sẽ đi tìm hiểu về các định dạng ở phần này.

4.3.2.1. Định dạng khóa bí mật

Khóa bí mật là một số nguyên dương 256 bit, tức nếu chúng ta viết dưới dạng nhị phân thì con số này sẽ chiếm 32 byte. Theo quy định của bitcoin, ta có hai định dạng có thể mã hóa khóa bí mật cho người dùng là WIF và WIF nén.

Với định dạng WIF thông thường, byte phiên bản cho mã hóa Base58Check có giá trị là 128 (hay 0x80 dưới hệ 16). Với tiền tố này, sau khi áp dụng mã hóa Base58Check, ký tự đầu tiên ta có sẽ là “5”.

Với định dạng WIF nén, quy trình làm sẽ tương tự như quy trình của WIF. Tuy nhiên, có một điểm khác biệt là ta cần phải thêm hậu tố 0x01 vào dữ liệu trước khi thực hiện quy trình mã hóa Base58Check. Bước cuối cùng sẽ như hình sau.



Hình 4.8: Mã hóa Base58Check cho khóa bí mật với định dạng WIF nén

Trong đó, Tổng kiểm tra được tính bằng cách lấy 4 byte đầu của giá trị có được sau khi áp dụng thuật toán SHA256 hai lần cho toàn bộ cụm dữ liệu bao gồm: byte phiên bản, khóa bí mật, và byte 0x01.

4.3.2.2. Định dạng khóa công khai

Tương tự như khóa bí mật, khóa công khai cũng tồn tại ở hai dạng: dạng thường và dạng nén.

Như trình bày lúc trước, khóa công khai là một điểm thuộc đường cong elliptic, tức khóa công khai sẽ có dạng (x, y) . Với dạng không nén, khóa công khai sẽ được bắt đầu bởi tiền tố 04, nối tiếp bởi hai con số 256 bit là tọa độ x và y . Với dạng nén, khóa công khai sẽ bắt đầu bởi tiền tố 02 hoặc 03, tiếp theo sẽ là tọa độ x , tọa độ y lúc này sẽ được lược bỏ. Chúng ta có thể lược bỏ giá trị y đi bởi vì khi đã có x , chúng ta có thể giải phương trình $y^2 \bmod p = (x^3 + 7) \bmod p$ để tính được y .

Một vấn đề nảy sinh ở đây sẽ là nếu ta nhìn phương trình, giá trị y sẽ được bình phương, tức khi tính lại giá trị y ta cần sử dụng phép căn bậc hai, đồng nghĩa với việc y có thể là dương hoặc âm. Nhìn lại Hình 4.1, thấy rằng y sẽ được phản xạ qua trục Ox , tức là tuy chúng ta có thể lược bỏ y nhưng chúng ta phải lưu lại dấu của y ban đầu. Theo như Andreas M. Antonopoulos có viết trong cuốn sách “Mastering Bitcoin: Programming the Open Blockchain” [12], khi chúng ta thực hiện tính toán trên đường cong elliptic trên tập hữu hạn có bậc nguyên tố p , giá trị của y có thể là chẵn hoặc lẻ, tương ứng với giá trị y dương hoặc âm. Vì lý do này, với khóa công khai dạng nén, chúng ta sẽ dùng tiền tố 02 với trường hợp y chẵn và 03 với trường hợp y lẻ.

Mục đích của dạng nén chính là để giảm thiểu số dung lượng cho blockchain. Nếu lưu toàn bộ dưới dạng không nén, mỗi khóa công khai sẽ cần tới 520 bit, bao gồm tiền tố, trong khi dùng dạng nén, mỗi khóa công khai chỉ cần 258 bit, bao gồm tiền tố. Với 50% dung lượng được tiết kiệm cho một khóa công khai, tích lũy dần qua thời gian là một con số rất lớn.

4.3.2.3. Lưu ý về định dạng khóa bí mật

Khi ta nói về định dạng WIF nén của khóa bí mật, thực sự thì định dạng không giúp ta tiết kiệm bộ nhớ, thay vào đó, định dạng này còn dùng hơn 1 byte so với định dạng không nén. Thực chất, lý do tồn tại của định dạng WIF nén là đảm bảo tính tương thích ngược cho nền tảng bitcoin.

Nhằm đảm bảo tính tương thích ngược khi định dạng nén của khóa công khai được đưa vào bitcoin, hai định dạng của khóa bí mật chỉ đơn giản mang ý nghĩa là khóa bí mật ở dạng WIF nào thì khóa công khai sinh ra phải ở dạng tương ứng. Tức nếu khóa bí mật thuộc dạng WIF nén, thì khóa công khai sinh ra từ khóa này sẽ là ở dạng nén, nếu khóa bí mật thuộc dạng WIF không nén thì khóa công khai sinh ra từ khóa bí mật này sẽ là dạng không nén.

4.4. Cài đặt khóa và địa chỉ sử dụng libbitcoin

Thư viện libbitcoin cung cấp cho chúng ta sẵn rất nhiều hàm để phục vụ việc lập trình bitcoin. Một chú ý trong quá trình sinh khóa bí mật, ta không cần phải thực sự phải chọn ngẫu nhiên một con số từ 1 tới $2^{256} - 1$ như trình bày bên trên. Thay vào đó, chúng ta có thể sinh ra một giá trị *seed* ngẫu nhiên với độ lớn bất kỳ, sau đó ta sẽ áp dụng thuật toán SHA256 vào seed đó. Vì SHA256 sẽ cho ra một chuỗi 256 bit, như vậy ta đã có được một con số từ 0 tới 2^{256} , ta chỉ cần kiểm tra một trường hợp duy nhất là giá bị băm có bằng 0 hay không.

```
1 bc::data_chunk seed(16); // 16 byte seed
2 bc::pseudo_random_fill(seed);
3
4 bc::ec_secret secret_key = bc::bitcoin_hash(seed); // SHA256
5
6 bc::ec_secret zero;
7 zero.fill(0);
8
9 while (secret_key == zero) {
10     bc::pseudo_random_fill(seed);
11     secret_key = bc::bitcoin_hash(seed);
12 }
```

Sau khi có khóa bí mật, áp dụng hàm `ec_multiply` sẽ giúp ta tính được khóa công khai. Sau đó, ta có thể sử dụng hàm `bitcoin_short_hash` để thực hiện quy trình HASH160.

```

1 bc::ec_compressed gen_point =
2     bc::base16_literal("0279BE667EF9DCBBAC55A06295CE870B07"
3                         "029BFCDB2DCE28D959F2815B16F81798");
4
5 bc::ec_compressed public_key(gen_point);
6 // nhân khóa bí mật cho điểm sinh
7 bc::ec_multiply(public_key, secret_key);
8 bc::short_hash address = bc::bitcoin_short_hash(public_key);

```

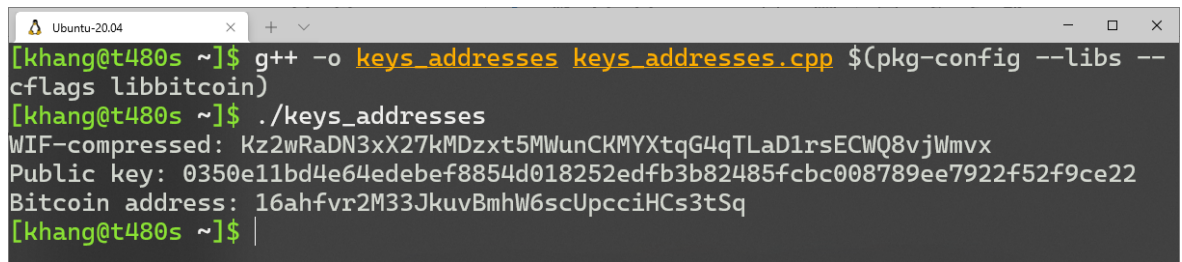
Cuối cùng là bước chuyển đổi sang định dạng Base58Check tương ứng cho địa chỉ và khóa bí mật, khóa công khai chỉ cần sử dụng mã hóa hệ 16.

```

1 // Base58Check cho địa chỉ
2 bc::data_chunk btc_address_raw;
3 btc_address_raw.push_back(0x00);
4 bc::extend_data(btc_address_raw, address);
5 bc::append_checksum(btc_address_raw);
6
7 std::string address_str = bc::encode_base58(btc_address_raw);
8
9 // Base58Check cho khóa bí mật với dạng WIF nén
10 bc::data_chunk wif_compressed_raw;
11 wif_compressed_raw.push_back(128);
12 bc::extend_data(wif_compressed_raw, secret_key);
13 wif_compressed_raw.push_back(0x01); // WIF nén
14 bc::append_checksum(wif_compressed_raw);
15
16 std::string wif_str = bc::encode_base58(wif_compressed_raw);
17
18 // Hex cho khóa công khai
19 std::string public_str = bc::encode_base16(public_key);

```

Nối các đoạn chương trình trên lại thành một chương trình hoàn chỉnh, khi thực thi thì kết quả có thể ra như sau:

A terminal window titled 'Ubuntu-20.04' showing the compilation and execution of a C++ program. The user runs 'g++ -o keys_addresses keys_addresses.cpp \$(pkg-config --libs --cflags libbitcoin)' and then './keys_addresses'. The output displays a WIF-compressed key, a public key, and a Bitcoin address.

```
[khang@t480s ~]$ g++ -o keys_addresses keys_addresses.cpp $(pkg-config --libs --cflags libbitcoin)
[khang@t480s ~]$ ./keys_addresses
WIF-compressed: Kz2wRaDN3xX27kMDzxt5MWunCKMYXtqG4qTLaD1rsECWQ8vjWmvx
Public key: 0350e11bd4e64edebef8854d018252edfb3b82485fcbc008789ee7922f52f9ce22
Bitcoin address: 16ahfvr2M33JkuvBmhW6scUpcciHCs3tSq
[khang@t480s ~]$
```

Hình 4.9: Kết quả thực thi chương trình mẫu sinh khóa và địa chỉ

Chương 5

Ví bitcoin

Với khả năng một người có thể sở hữu nhiều cặp khóa, việc người dùng tự quản lý hết số lượng khóa của họ là một điều cực kỳ khó khăn. “Ví” có thể hiểu là một ứng dụng quản lý thay cho người dùng các khóa, địa chỉ, và giao dịch. Ngoài ra, “ví” còn có thể hiểu theo một ý nghĩa khác là cấu trúc dùng để lưu trữ khóa. Chương 5 sẽ đi qua về ba công nghệ ví cơ bản, tìm hiểu những khả năng của loại ví đó. Từ đó, chúng ta sẽ chọn ra một loại công nghệ ví tốt nhất, và đi vào chi tiết về quá trình tạo lập ví.

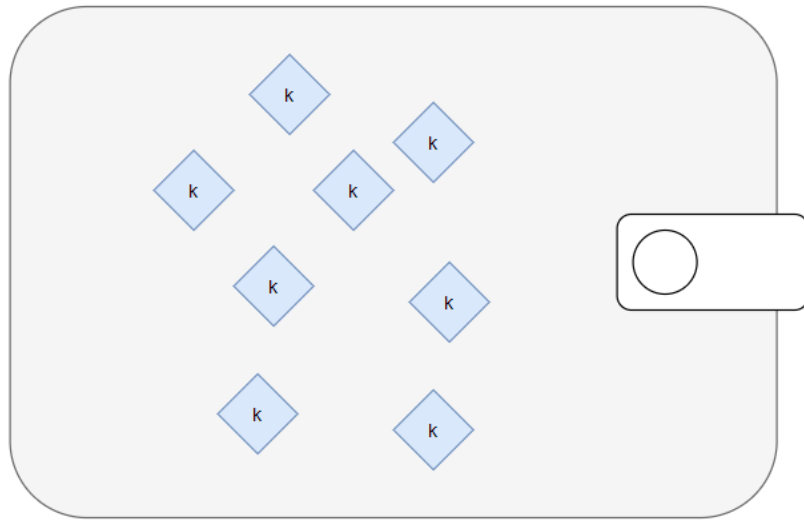
5.1. Tổng quan các công nghệ ví

Khi nhắc tới ví, một người có thể nghĩ ngay tới việc đó là một ứng dụng lưu trữ tiền bitcoin mà mình đang sở hữu. Tuy nhiên, ví bitcoin không thực sự lưu số bitcoin mình sở hữu, thay vào đó, ví bitcoin chỉ lưu khóa mà thôi, còn số bitcoin mình sở hữu đã được lưu trữ thông qua danh sách các giao dịch trên blockchain.

5.1.1. Ví ngẫu nhiên

Loại công nghệ ví sơ khai nhất có thể nói là ví ngẫu nhiên (*nondeterministic wallet*). Công nghệ ví này có thể thấy sử dụng trong những phiên bản đầu tiên của Bitcoin Core. Vì tính chất sinh ngẫu nhiên của ví, ví cần phải lưu lại toàn bộ các khóa đã được tạo ra và lưu trữ một cách an toàn và thường xuyên.

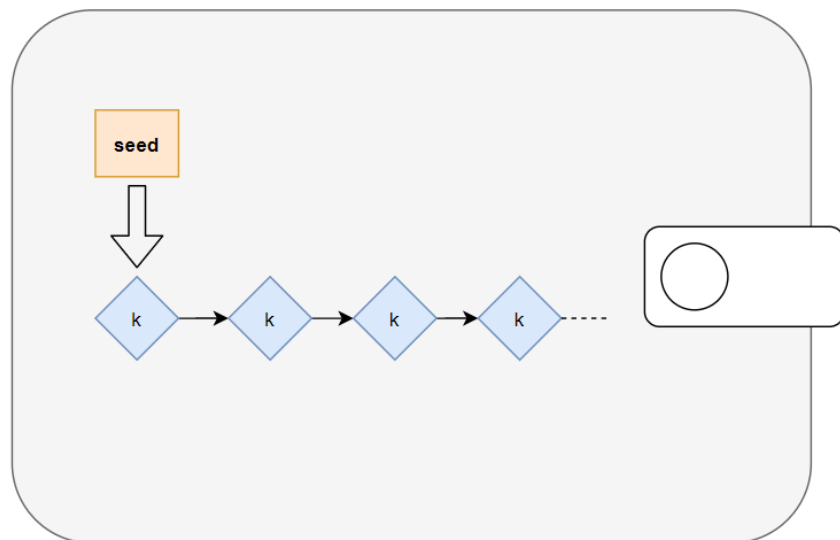
Bởi vì tính chất rời rạc này, nếu chúng ta làm mất bản lưu trữ dữ liệu của ví, toàn bộ số bitcoin của chúng ta không còn lấy lại được nữa. Tuy Bitcoin Core đã từng sử dụng ví ngẫu nhiên, hiện tại Bitcoin Core đã chuyển sang dùng ví HD và các nhà phát triển Bitcoin Core không khuyến khích sử dụng loại ví này.



Hình 5.1: Hình mô phỏng ví ngẫu nhiên

5.1.2. Ví xác định

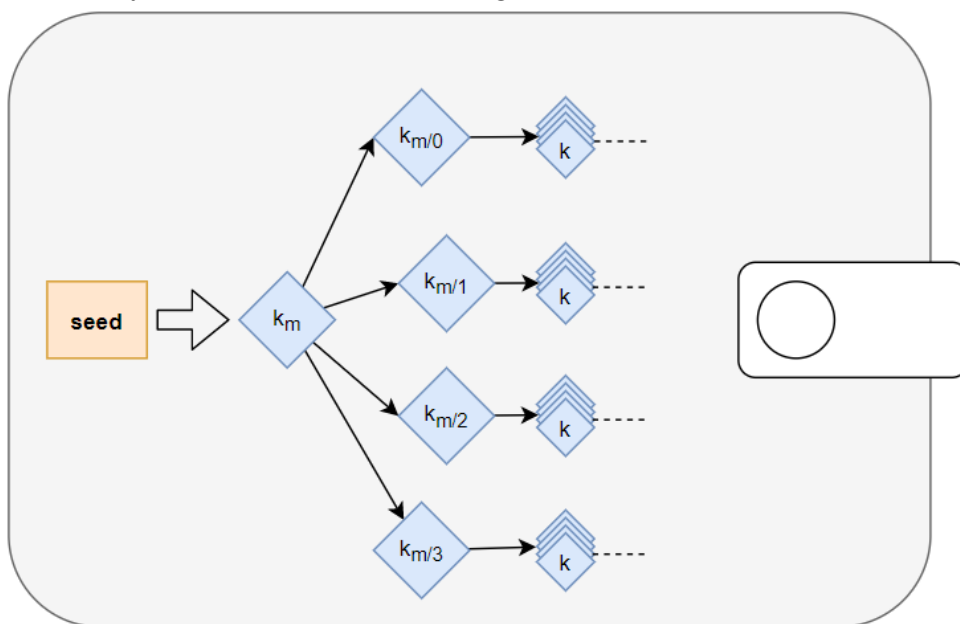
Ví xác định (*deterministic wallet*) là một kiến trúc ví sử dụng một giá trị *seed* để sinh ra khóa đầu tiên. Sau đó, để tạo được khóa tiếp theo, hàm dẫn xuất khóa sẽ được sử dụng lên khóa mới nhất hiện tại. Ví này giải quyết được vấn đề đảm bảo tính an toàn của ví bởi vì chỉ cần chúng ta lưu trữ được giá trị *seed* là chúng ta đã có thể xây dựng lại toàn bộ ví một cách dễ dàng.



Hình 5.2: Hình mô phỏng ví xác định

5.1.3. Ví HD

Cấu trúc ví thứ 3 cũng là loại cấu trúc ví tốt nhất của chúng ta là ví xác định có thứ bậc (*Hierarchical deterministic wallet*, hay *HD wallet*) [15]. Tên của ví đã nói lên tất cả, đây là loại ví sử dụng tính chất của ví xác định là từ giá trị seed để tạo ra khóa đầu tiên. Tuy nhiên, từ một khóa, chúng ta có thể sinh ra nhiều nhánh khác nhau.



Hình 5.3: Hình mô phỏng ví HD

Ví HD bên cạnh việc quản lý các khóa tốt hơn ví ngẫu nhiên, loại ví này còn có thể giúp chúng ta quản lý các khóa một cách gọn gàng hơn với việc sử dụng từng nhánh khóa cho từng ý nghĩa khác nhau. Nếu người dùng mong muốn, người dùng có thể sử dụng ví HD để sinh ra một khóa công khai mới để nhận bitcoin mà không cần phải thông qua khóa bí mật tương ứng. Khả năng này khiến cho ví HD thực sự mạnh mẽ nếu người dùng thực hiện giao dịch trên những máy chủ hoặc hệ thống mạng không an toàn mà vẫn không phải lo lắng về vấn đề khóa bí mật bị mất mát.

5.2. Chi tiết về ví HD

Ví HD là một công nghệ ví quản lý giúp người dùng khóa và địa chỉ rất tốt. Việc áp dụng những giao thức tiêu chuẩn để tạo ví sẽ giúp cho ví HD càng trở nên tốt hơn.

5.2.1. Mã mnemonic

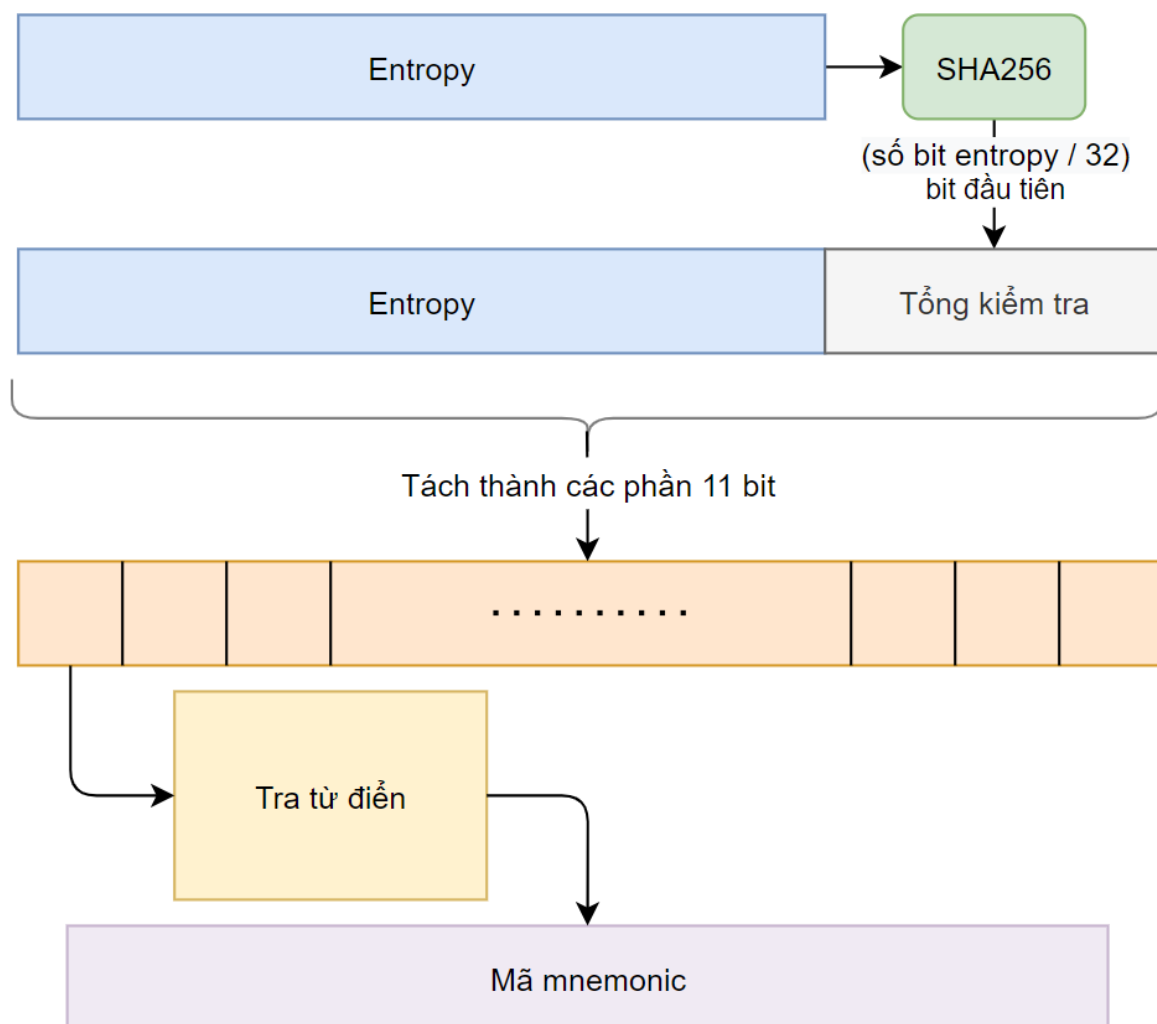
Mã mnemonic, lần đầu được định nghĩa ở BIP-0039 [16], là một tập hợp các từ tiếng Anh dễ nhớ, tượng trưng cho một giá trị seed được dùng để tạo ra một ví HD hoàn chỉnh. Một ví sử dụng mã mnemonic khi lần đầu khởi tạo ví sẽ tạo ra cho người dùng một chuỗi từ 12 cho đến 24 từ. Chuỗi đó là một mã dự phòng để tái tạo lại ví với toàn bộ các khóa. Mã mnemonic được tạo ra bởi vì một danh sách các từ tiếng Anh rất dễ để sao lưu và ghi chép hơn giá trị seed.

Theo như Andreas M. Antonopoulos [12], BIP-0039 ở thời điểm hiện tại đã đạt được một sự đón nhận lớn trong cộng đồng bitcoin và được xem là một tiêu chuẩn mặc định, chiến thắng một tiêu chuẩn khác không tương thích với BIP-0039 do nhà phát triển ví Electrum tạo ra.

5.2.1.1. Tạo mã mnemonic

Quy trình tạo mã mnemonic được diễn ra như sau:

1. Tạo ra một entropy từ 128 đến 256 bit;
2. Thêm vào cuối entropy một tổng kiểm tra bằng cách lấy $\left(\frac{\text{số bit entropy}}{32}\right)$ bit đầu tiên trong mã băm SHA256 của entropy;
3. Chia chuỗi trên thành các phần 11 bit;
4. Ánh xạ từng phần đó vào một từ điển định nghĩa sẵn 2048 từ;
5. Danh sách các từ vừa được ánh xạ là mã mnemonic.



Hình 5.4: Quy trình tạo mã mnemonic từ entropy

Bảng bên dưới sẽ thể hiện cụ thể độ lớn có thể có của các thành phần trong quy trình tạo mã mnemonic.

Bảng 5.1: Độ lớn của các thành phần trong quá trình tạo mã mnemonic

Số bit entropy	Số bit tổng kiểm tra	Số lượng từ trong mã mnemonic
128	4	12
160	5	15
192	6	18
224	7	21
256	8	24

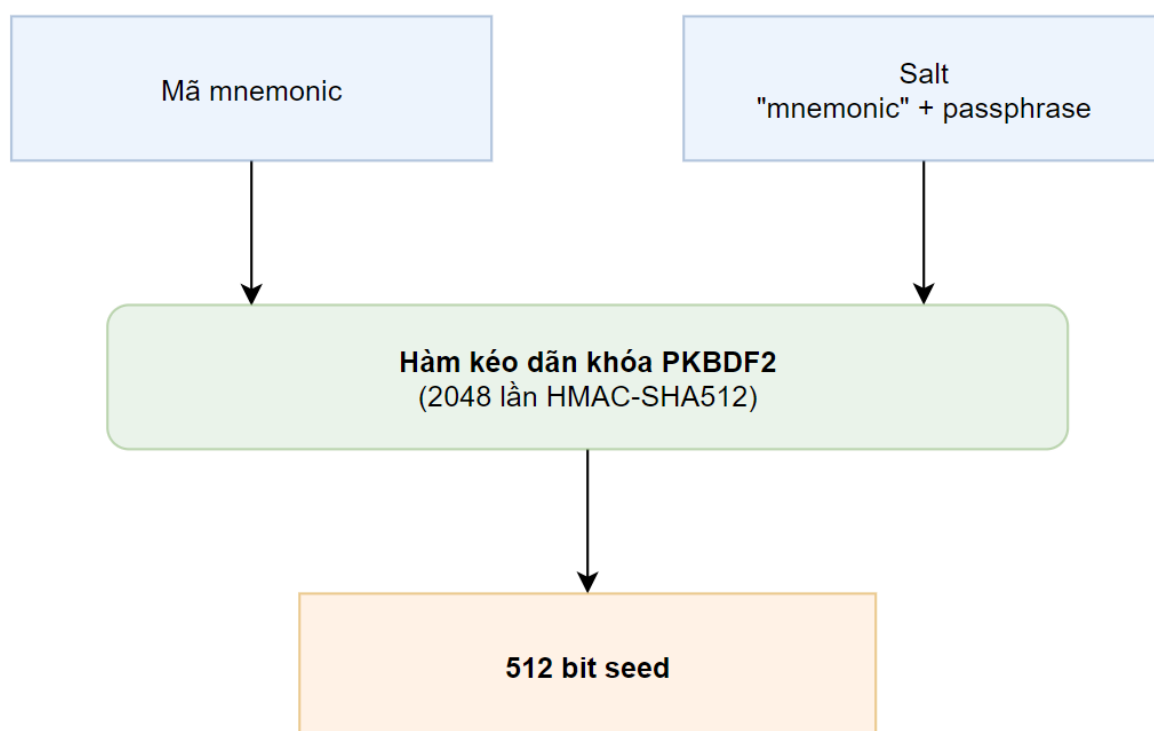
5.2.1.2. Cấu tạo seed từ mã mnemonic

Như đã trình bày bên trên, mã mnemonic là cách biểu diễn entropy dưới dạng một tập hợp các từ trong từ điển. Entropy này sau đó sẽ được dùng để tạo ra một seed dài hơn, sử dụng *hàm kéo dẫn khóa (key-stretching function)* PBKDF2. Seed này sau đó sẽ được dùng để tạo thành ví xác định và dẫn xuất khóa.

Chúng ta sẽ cung cấp cho hàm kéo dẫn khóa PBKDF2 hai tham số:

1. Tham số đầu tiên là mã mnemonic;
2. Tham số thứ hai được gọi là *salt* bao gồm hai thành phần: phần đầu tiên là xâu “mnemonic” được nối bởi một mật khẩu do người dùng cung cấp (mật khẩu này không nhất thiết phải tồn tại). Nếu mật khẩu không tồn tại, một xâu rỗng sẽ được sử dụng.

Hàm PBKDF2 sẽ sử dụng hai tham số trên và chạy qua 2048 vòng lặp, với thuật toán băm tương ứng là HMAC-SHA512 để cho ra một dãy kết quả 512 bit. Đây sẽ là seed của chúng ta.



Hình 5.5: Quy trình tạo seed từ mã mnemonic

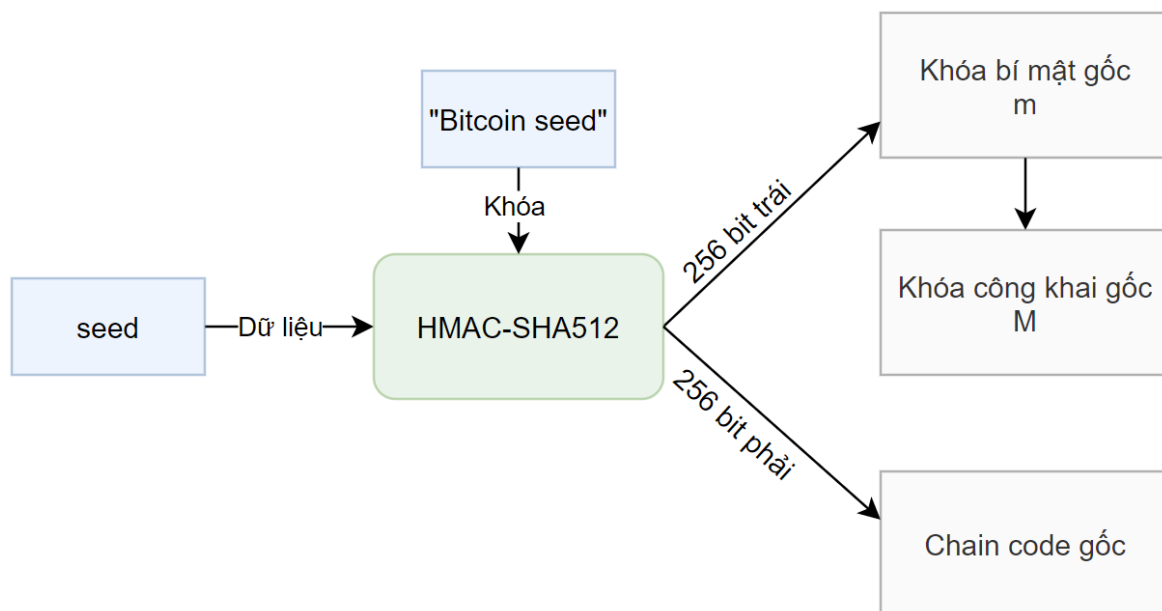
5.2.2. Dẫn xuất khóa

Như đã trình bày, ví HD sẽ được tạo thành từ một giá trị seed gốc, giá trị này có thể có các độ lớn 128 bit, 256 bit, hoặc 512 bit. Thông thường, seed của ví HD sẽ được xây dựng dựa trên mã mnemonic được trình bày bên trên.

Tất cả các khóa của ví HD đều được dẫn xuất một cách xác định dựa vào seed. Vì lý do này, ta có thể dễ dàng xây dựng lại toàn bộ các khóa mà không xảy ra tình trạng mất mát, trong khi chỉ cần sao lưu duy nhất mã mnemonic và mật khẩu nếu có.

Đầu tiên, ta cần bàn về cách tạo ra khóa gốc từ seed:

1. Sử dụng seed, ta đưa seed vào hàm băm HMAC-SHA512 với khóa là chuỗi “Bitcoin seed” [15]. Đầu ra của hàm băm này là một mã băm 512 bit;
2. Sử dụng mã băm 512 bit trên, 256 bit bên trái được lấy làm giá trị của khóa bí mật gốc (ký hiệu m), 256 bit bên phải còn lại sẽ được lưu vào một giá trị có tên là *chain code*⁷ gốc (*master chain code*).



Hình 5.6: Quy trình tạo khóa gốc từ seed

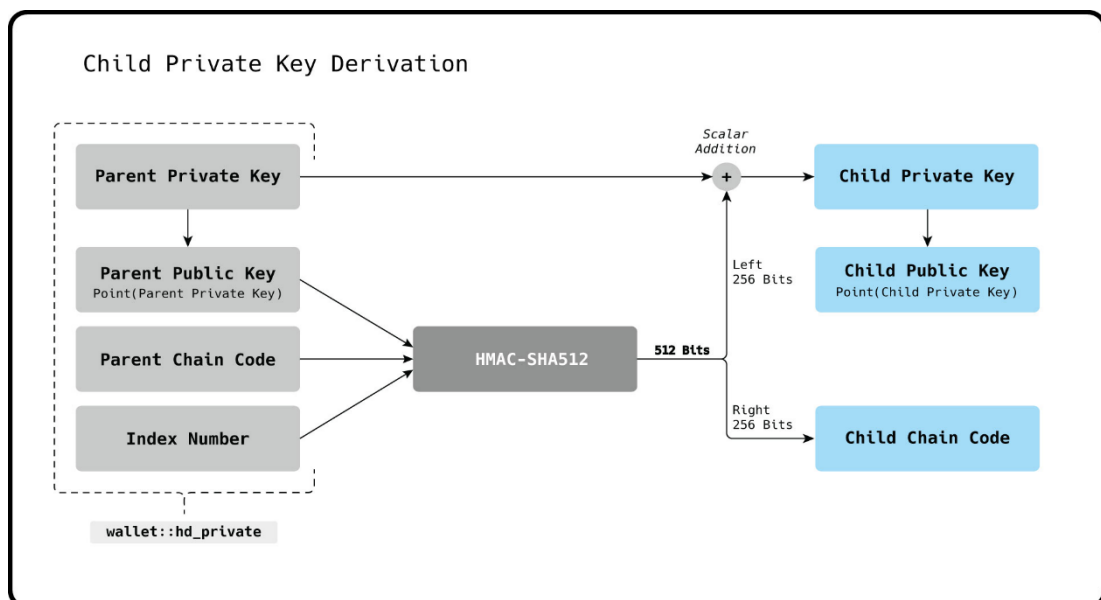
⁷ Chain code là một giá trị mà nó được sử dụng như một entropy trong quá trình dẫn xuất khóa con từ khóa tương ứng.

Sau khi có được khóa bí mật gốc m , ta có thể dễ dàng tính được khóa công khai gốc (ký hiệu M) bằng cách lấy khóa bí mật nhân với điểm G như công thức 4.3.

5.2.2.1. Dẫn xuất khóa bí mật

Có được khóa gốc, chúng ta có thể thực hiện công việc dẫn xuất những khóa tiếp theo như sau:

1. Từ khóa bí mật cha, ta sẽ sử dụng khóa để tạo ra khóa công khai tương ứng;
2. Với khóa công khai vừa tạo, ta thêm một byte hậu tố là số thứ tự (tính từ 0) của khóa con; sử dụng cụm giá trị này làm dữ liệu cho hàm băm;
3. Sử dụng giá trị chain code của khóa cha làm khóa của hàm băm, áp dụng hàm HMAC-SHA512 với dữ liệu và khóa tương ứng, ta sẽ có được một mã băm 512 bit;
4. Từ mã băm 512 bit trên, ta lấy giá trị của khóa bí mật cha cộng với 256 bit trái sẽ được giá trị của khóa bí mật con, 256 bit phải làm chain code cho khóa con.



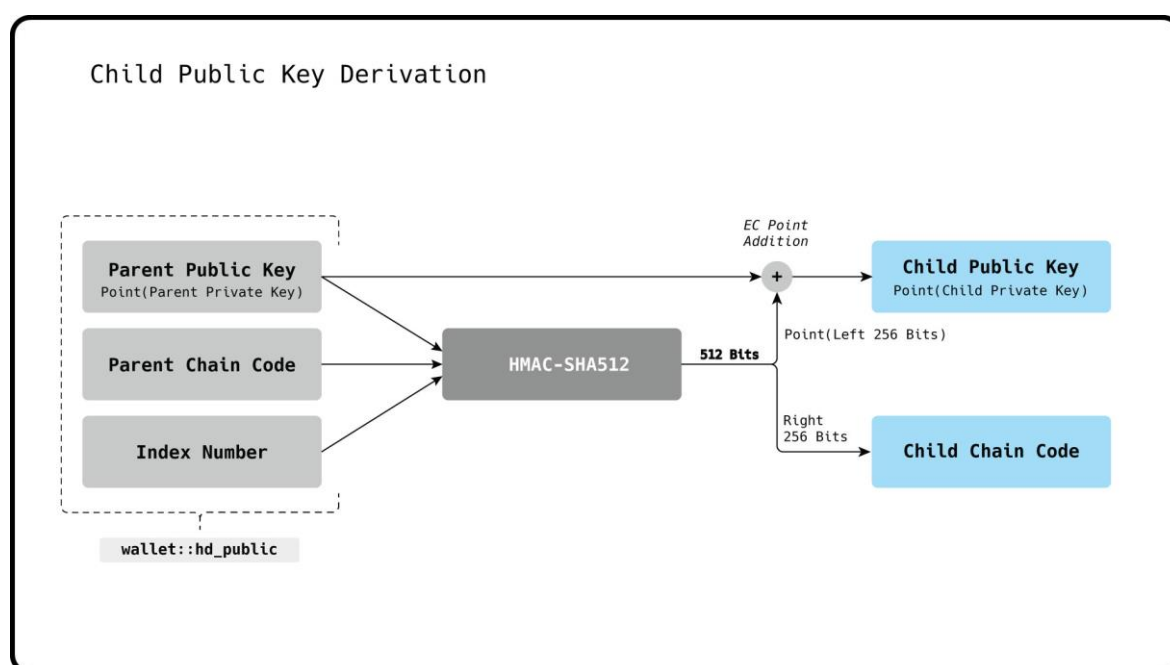
Hình 5.7: Quy trình dẫn xuất khóa bí mật của ví HD

(Nguồn: <https://github.com/libbitcoin/libbitcoin-system/wiki/Addresses-and-HD-Wallets>)

5.2.2.2. Dẫn xuất khóa công khai

Quy trình dẫn xuất khóa công khai được diễn ra như sau:

1. Tương tự như dẫn xuất khóa bí mật, đầu tiên ta sẽ sử dụng khóa công khai cha, thêm một byte hậu tố là số thứ tự của khóa con, và sử dụng cụm giá trị làm giá trị cho hàm băm;
2. Áp dụng hàm băm HMAC-SHA512 với cụm dữ liệu bên trên và với khóa là giá trị chain code của khóa cha. Đầu ra sẽ là một mã băm 512 bit;
3. Từ mã băm 512 bit bên trên, lấy 256 bit bên thực hiện phép cộng trên đường cong elliptic với giá trị của khóa công khai cha để tạo thành giá trị của khóa công khai con, còn 256 bit phải sử dụng làm chain code cho khóa con tương tự như quá trình dẫn xuất khóa bí mật.



Hình 5.8: Quy trình dẫn xuất khóa công khai của ví HD

(Nguồn: <https://github.com/libbitcoin/libbitcoin-system/wiki/Addresses-and-HD-Wallets>)

5.2.2.3. Khóa cứng

Khái niệm tiếp theo chúng ta tìm hiểu sẽ là khái niệm về *khóa cứng* (*hardened key*). Khái niệm khóa cứng được áp dụng cho cả khóa bí mật và khóa công khai. Khóa cứng là một dạng khóa mà chỉ có khóa bí mật mới có thể dẫn xuất được một khóa con cứng. Khóa con này có thể là khóa bí mật hoặc khóa công khai.

5.2.3. Đường dẫn khóa

5.2.3.1. Quy tắc đặt tên đường dẫn khóa

Đường dẫn của một khóa thuộc khóa HD là quá trình dẫn xuất đến được khóa đó. Đường dẫn của khóa sẽ giúp chúng ta có thể dễ dàng thấy được kiến trúc tổng quan của ví.

Ký tự bắt đầu sẽ là ‘*m*’ hoặc ‘*M*’, tương ứng cho khóa bí mật gốc hoặc khóa công khai gốc. Các tầng dẫn xuất được phân tách lẫn nhau bởi một ký tự ‘/’, tiếp theo sẽ là số thứ tự của khóa con tương ứng. Sau số thứ tự, nếu khóa con đó là một khóa cứng, chúng ta thêm ký tự ‘’ (dấu nháy đơn) vào ngay sau số thứ tự đó.

Bảng 5.2: Một số ví dụ về đường dẫn khóa

Định danh	Giải thích
m/0	Khóa bí mật con đầu tiên của khóa bí mật gốc
m/0’	Khóa bí mật con cứng đầu tiên của khóa bí mật gốc
m/1’/0/12	Khóa bí mật con thứ 13 của khóa con đầu tiên của khóa con cứng thứ 2 của khóa bí mật
M/15/1/3	Khóa công khai con thứ 16 của khóa công khai gốc

5.2.3.2. BIP-0044

BIP-0044 được tạo ra nhằm lập nên một quy chuẩn của quá trình dẫn xuất khóa và mở rộng khả năng hỗ trợ đa tài khoản cho ví HD [17]. Đường dẫn dẫn xuất khóa của BIP-0044 được quy định như sau:

```
m / purpose' / coin_type' / account' / change / address_index
```

Với các thành phần:

- `m`: khóa bí mật gốc;
- `purpose`: trường này được cố định là đường dẫn 44'. Theo như BIP-0044 [17], trường này áp dụng quy tắc của BIP-0043⁸;
- `coin_type`: một seed gốc có thể được sử dụng cho nhiều loại tiền ảo khác, không chỉ riêng mình Bitcoin;
- `account`: số thứ tự tài khoản;
- `change`: 0 nếu là tài khoản sử dụng cho việc nhận tiền (*external chain*), 1 nếu là tài khoản sử dụng cho việc thổi tiền (*internal chain*);
- `address_index`: số thứ tự tài khoản theo đường dẫn tương ứng.

Quy trình khám phá tài khoản

Khi ví được khởi tạo, ví sẽ thực hiện quá trình khám toàn bộ các địa chỉ có thể.

Quy trình khám phá được quy định như sau [17]:

1. Dẫn xuất tài khoản đầu tiên (`account = 0`);
2. Dẫn xuất tài khoản nhận tiền;
3. Duyệt tài khoản nhận tiền với giới hạn được quy định;
4. Nếu không có giao dịch nào tìm thấy, dừng khám phá;
5. Nếu tìm thấy ít nhất một giao dịch, tăng số thứ tự tài khoản và quay về bước 1.

Trong quá trình khám phá tài khoản, nếu ta tìm thấy liên tục 20 địa chỉ chưa được dùng thì ta sẽ coi như tài khoản đó chưa được dùng và dừng việc tìm kiếm tài khoản.

⁸ BIP-0043: Purpose Field for Deterministic Wallets. Nguồn: https://en.bitcoin.it/wiki/BIP_0043.

Chương 6

Giao dịch bitcoin

Một đồng tiền tệ không thể nào hoạt động nếu thiếu đi việc giao dịch. Bitcoin sử dụng khóa bí mật và khóa công khai để thực hiện các thao tác gửi và nhận. Chương này sẽ đi vào tìm hiểu về giao dịch bitcoin được cấu tạo như thế nào.

6.1. Tổng quan về giao dịch bitcoin

Ở Chương 2, chúng ta đã làm quen tới tình huống mua đồ ngoài cửa hàng tiện lợi sử dụng bitcoin. Sau đó, chúng ta dùng ứng dụng khám phá blockchain để xem lại giao dịch mua đồ này ở Hình 2.1.

Nếu chúng ta thực hiện xem chi tiết giao dịch thông qua Bitcoin Core (Hình 3.6), chúng ta có thể thấy được cấu trúc tổng quan của một giao dịch có cấu trúc như thế nào.

Để ý rằng cấu trúc giao dịch nhìn tương đối phức tạp. Ở phần tiếp theo, chúng ta sẽ đi vào tìm hiểu chi tiết và giải mã các thành phần của một giao dịch bitcoin.

6.2. Đầu vào và đầu ra giao dịch

Một giao dịch bitcoin sẽ có đầu vào và đầu ra. Đầu vào sẽ như số tiền mình trả cho người khác, còn đầu ra quy định ai là người nhận số tiền đó và bao nhiêu (và có bao gồm cả tiền thừa). Mỗi giao dịch sẽ có một mã định danh riêng, có tên là *transaction ID (TXID)*. Sử dụng những công cụ khám phá blockchain, nếu chúng ta có được TXID của một giao dịch thì chúng ta có thể sử dụng mã định danh này để xem lại nội dung chi tiết của một giao dịch.

Khi chúng ta nói về việc nhận bitcoin, vì bitcoin không tồn tại, việc này không phải là mình nhận một lượng bitcoin, mà là mình nhận được một *đầu ra giao dịch chưa sử dụng (unspent transaction output – UTXO)*, mà UTXO này có thể được sử dụng bởi một cặp khóa do mình sở hữu. Mỗi giao dịch sẽ sử dụng một lượng UTXO

nhất định, và đồng thời cũng tạo ra một lượng UTXO nhất định. Tương tự cho việc gửi bitcoin, công việc này sẽ sử dụng những UTXO của mình đang có để tạo ra một UTXO mới, mà người nhận có thể dùng nó để thực hiện các giao dịch tiếp theo. Ví bitcoin thực hiện công việc tính toán số dư bằng cách tính tổng giá trị của toàn bộ UTXO mình đang nắm giữ. Các UTXO này có khả năng sẽ bị rải rác ở nhiều địa chỉ mà mình đang sở hữu, và mình có thể dùng với khóa tương ứng cho từng địa chỉ.

Đầu vào của giao dịch sẽ là một danh sách các UTXO mà người dùng đó đang sở hữu. Trong giao dịch bitcoin, giá trị của một UTXO không thể bị chia nhỏ, dẫn tới khả năng chúng ta sẽ trả dư so với nhu cầu chi tiêu của chúng ta. Vì lý do này, có một dạng đầu ra đặc biệt là dạng tiền thừa. Với dạng đầu ra này, số tiền thừa còn lại sẽ được gửi về một địa chỉ mà mình sở hữu, thay vì cho người nhận. Một đầu ra của giao dịch sẽ tương trưng như một UTXO dành cho người nhận tương ứng. Số lượng đầu ra và đầu vào tối đa của giao dịch sẽ được biểu diễn dưới dạng số nguyên dương 9 byte, tuy nhiên, sau khi tạo ra thì chúng ta không thể chia nhỏ chúng.

Vấn đề trên giống như việc chúng ta thanh toán các giao dịch sử dụng tiền mặt. Một UTXO giống như một tờ tiền trong ví chúng ta. Để trả đủ số tiền mong muốn, chúng ta có thể gom nhiều UTXO lại với nhau để giá trị đạt đủ giá trị chi trả, giống như việc sử dụng nhiều tờ tiền cùng một lúc. Tuy nhiên, chúng ta không thể xé nhỏ một tờ tiền có mệnh giá lớn hơn thành nhiều tờ mệnh giá nhỏ hơn được, chúng ta phải dùng toàn bộ giá trị. Nếu giá trị chúng ta sử dụng bị dư, chúng ta sẽ nhận lại tiền thừa.

Vì đầu ra đơn giản hơn, và đầu vào của một giao dịch có phụ thuộc vào đầu ra nên chúng ta sẽ đi tìm hiểu về đầu ra trước.

6.2.1. Đầu ra giao dịch

Một giao dịch bitcoin luôn luôn phải có đầu ra và nội dung sẽ được ghi lại trên blockchain. Gần như mọi đầu ra giao dịch đều tạo ra một lượng bitcoin có thể sử dụng dưới dạng một UTXO. Toàn bộ UTXO sẽ được ghi chép lại bởi các nút đầy đủ.

Đầu ra giao dịch gồm hai phần:

- Giá trị: Giá trị bitcoin của đầu ra dưới dạng *satoshi*⁹;
- Kịch bản khóa (*locking script*): Một điều kiện để sử dụng được đầu ra này. Locking script có các tên gọi như là *witness script*, hay *scriptPubKey*.

Trong giá trị trả về của nội dung giao dịch mua đồ, danh sách đầu ra được định nghĩa trong phần *vout*:

```
"vout": [
  {
    "value": 0.00015000,
    "n": 0,
    "scriptPubkey": {
      "asm": "OP_DUP OP_HASH160 4538024caf2fd1decb379dee704ccc488a531394
OP_EQUALVERIFY OP_CHECKSIG",
      "hex": "76a9144538024caf2fd1decb379dee704ccc488a53139488ac"
    },
    "reqSigs": 1,
    "type": "pubkeyhash",
    "addresses": [
      "mmpx1rzE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ"
    ]
  },
  {
    "value": 0.00084500,
    "n": 1,
    "scriptPubkey": {
      "asm": "OP_DUP OP_HASH160 746ca5f027d7abb2351ef4501f85282c2ecf3c6a
OP_EQUALVERIFY OP_CHECKSIG",
      "hex": "76a914746ca5fe27d7abb2351ef4501f85282c2ecf3c6a88ac"
    },
    "reqsigs": 1,
    "type": "pubkeyhash",
    "addresses": [
      "mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epv"
    ]
  }
]
```

Nhìn vào đoạn giá trị trên, chúng ta có thể thấy rằng giao dịch này có hai đầu ra. Mỗi đầu ra bao gồm *value* và *scriptPubKey* như đã trình bày bên trên. Ta còn có thêm

⁹ 1 satoshi = 10⁻⁸ bitcoin

một số trường khác được trả về kèm theo hai trường trên, các trường này là do Bitcoin Core tự thêm vào cho chúng ta, giúp cho chúng ta xử lý và đọc nội dung của giao dịch dễ dàng hơn.

Xâu chuỗi đầu ra giao dịch

Để tiết kiệm băng thông truyền tải và dễ dàng xử lý trong lúc lập trình, chúng ta cần phải *xâu chuỗi hóa* (*serialize*) đầu ra giao dịch. Định dạng xâu chuỗi của đầu ra giao dịch được định nghĩa như sau:

Bảng 6.1: Định dạng xâu chuỗi của một đầu ra giao dịch

Trường	Độ lớn (byte)
Số tiền theo đơn vị satoshi dưới dạng little-endian	8
Độ lớn locking script	Tùy biến từ 1 đến 9^{10}
Locking script	Tùy biến

Nhìn lại giao dịch dưới định dạng thô (đã được mã hóa hệ 16), ta có thể thấy phần dữ liệu chứa nội dung giao dịch thuộc phần in đậm:

```
010000000130d73640da0ed6ba686a50d5b7c4014eb16
8f90827c0104a96da74720a7157c1000000006b483045
022100deca2ec2e58599d99bf86528d3c13efa3b4d886
04248ca87e430e768dd4c524b022030e3a7dcfc8e0ba3
652cecee336987f87f89a1d257b25e18f68d4477eff60
8a00121026a0c583ad692cd43458154898b4ed3de261b
e6826d249af48397f171fcbe424cfffffffff02983a000
000000001976a9144538024caf2fd1dec379dee704c
cc488a53139488ac144a010000000001976a914746ca
5f027d7abb2351ef4501f85282c2ecf3c6a88ac00000000
```

Thấy trắng trước giao dịch đầu tiên ta có thêm một byte có giá trị là **02**, đây là byte cho biết giao dịch của chúng ta có bao nhiêu đầu ra tất cả, trường hợp này là 2. Bảng bên dưới là giải mã cho đoạn in đậm trên.

¹⁰ Số nguyên có độ lớn tùy biến: https://en.bitcoin.it/wiki/Protocol_documentation#Variable_length_integer.

Bảng 6.2: Giải mã đầu ra giao dịch mẫu

Giá trị	Giải thích
02	Có 2 đầu ra
983a000000000000	Đầu ra có giá trị 15000 satoshi
19	Locking script có độ lớn 25 byte
76a91445... (đủ 25 byte)	Locking script của đầu ra này
144a010000000000	Đầu ra có giá trị 84500 satoshi
19	Locking script có độ lớn 25 byte
76a91474... (đủ 25 byte)	Locking script của đầu ra này

6.2.2. Đầu vào giao dịch

Một giao dịch bitcoin được thực hiện dựa trên các UTXO. Đầu vào của giao dịch là phần xác định những UTXO nào sẽ được sử dụng và chứng minh rằng mình là người sở hữu quyền sử dụng UTXO tương ứng.

Đầu vào giao dịch bao gồm các phần sau:

- TXID chứa UTXO mình muốn sử dụng;
- Số thứ tự đầu ra tương ứng với UTXO mình muốn sử dụng trong TXID cung cấp bên trên;
- Kịch bản mở khóa (*unlocking script*): Một chuỗi các thao tác dùng để chứng minh tính sở hữu UTXO tương ứng.

Kịch bản mở khóa còn có tên gọi khác là *scriptSig*;

- Một số nguyên có tên là *sequence* là phiên bản giao dịch, giá trị thường thấy sẽ là 0xFFFFFFFF, hiện tại chưa có tính năng sử dụng.

Trong giá trị trả về của nội dung giao dịch mua đồ, danh sách đầu vào được định nghĩa ở phần vin:

```

"vin": [
  {
    "txid": "c157710a7274da964a10c02708f968b14e01c4b7d5506a68bad60eda4036d730",
    "vout": 0,
    "scriptSig": {
      "asm": "3045022100deca2ec2e58599d99bf86520d3c13efa3b4d88604248ca07e430e768dd4c524b022030e3a7dcfc8e0ba3652cecee336987f87f89a1d257b25e18f68d4477eff608a0 [ALL] 026a0c583ad692cd4345815d898b4ed3de261be6826d249af40397f171fcbe424c",
      "hex": "483045022100deca2ec2e58599d99bf86520d3c13efa3b4d80604248ca07e436e768dd4c524b022030e3a7dcfc8e0ba3652cecee336987f87f89a1d257b25e18f68d4477eff608a00121026a0c583ad692cd4345815d898b4ed3de261be6826d249af40397f171fcbe424c"
    },
    "sequence": 4294967295
  }
]

```

Nhìn vào đoạn mã trên, ta có thể thấy rằng đoạn unlocking script rất dài., nội dung chi tiết về script sẽ được tìm hiểu vào phần sau.

Xâu chuỗi đầu vào giao dịch

Định dạng chuỗi của đầu vào giao dịch được định nghĩa như sau:

Bảng 6.3: Định dạng chuỗi của đầu ra giao dịch

Trường	Độ lớn (byte)
TXID cũ	32
Số thứ tự đầu ra dưới dạng little-endian	4
Độ lớn unlocking script	Tùy biến từ 1 đến 9
Unlocking script	Tùy biến
Giá trị sequence	4

Phần được in đậm bên dưới là nội dung đầu vào của giao dịch:


```
010000000130d73640da0ed6ba686a50d5b7c4014eb16
8f90827c0104a96da74720a7157c1000000006b483045
022100deca2ec2e58599d99bf86528d3c13efa3b4d886
04248ca87e430e768dd4c524b022030e3a7dcfc8e0ba3
652cecee336987f87f89a1d257b25e18f68d4477eff60
8a00121026a0c583ad692cd43458154898b4ed3de261b
e6826d249af48397f171fcbe424cfffffffff02983a000
000000001976a9144538024caf2fd1dec379dee704c
cc488a53139488ac144a010000000001976a914746ca
5f027d7abb2351ef4501f85282c2ecf3c6a88ac00000000
```

Tương tự như đầu ra, đầu phần dữ liệu vào sẽ có một byte 01 có ý nghĩa là có 1 đầu vào. Bên dưới là bảng giải mã.

Bảng 6.4: Giải mã đầu vào giao dịch mẫu

Giá trị	Giải thích
01	Giao dịch có 1 đầu vào
30d736... (đủ 32 byte)	TXID chứa UTXO muốn sử dụng
00000000	Sử dụng đầu ra 0 của TXID trên
6b	Unlocking script có độ lớn 107 byte
483045... (đủ 107 byte)	Unlocking script
ffffffffff	Giá trị sequence

6.2.3. Xâu chuỗi giao dịch

Quá trình xâu chuỗi giao dịch thực sự tương đối dễ dàng sau khi biết được cách thức xâu chuỗi đầu vào và đầu ra giao dịch. Nếu nhìn vào giao dịch thô, ta chỉ còn hai phần còn lại của giao dịch là hai thành phần chưa được giải thích nằm ở đầu và cuối của giao dịch thô. Hai giá trị đó được giải thích ở bảng bên dưới.

Bảng 6.5: Giải thích phiên bản và thời gian khóa trong giao dịch

Tên	Giá trị	Giải thích
Phiên bản	01000000	Phiên bản giao dịch được ghi dưới dạng little-endian
Thời gian khóa (lock time)	00000000	Giao dịch chỉ được xử lý khi đạt đến thời điểm này (định dạng little-endian)

Tính cho đến thời điểm hiện tại, ta có hai phiên bản giao dịch, tương ứng cho giá trị phiên bản là 1 và 2. Phiên bản 1 là phiên bản đang được trình bày, phiên bản 2 có ý nghĩa là giao dịch có sử dụng BIP-0068¹¹.

Một số giá trị có thể cho trường thời gian khóa như sau:

- Giá trị bằng 0: Giao dịch không bị khóa và được xử lý ngay khi có thể;
- Giá trị nhỏ hơn 500,000,000: Giao dịch bị khóa cho đến block tương ứng;
- Giá trị từ 500,000,000 trở lên: Giao dịch bị khóa cho đến khi giờ UNIX đạt đến thời điểm này.

Với hai trường hợp sau, giá trị sequence của các đầu vào giao dịch phải khác giá trị mặc định là 0xFFFFFFFF để trường thời gian khóa có hiệu lực.

Tổng hợp lại, chuỗi một giao dịch bao gồm các phần sau:

Bảng 6.6: Các thành phần cần có trong một chuỗi giao dịch

Tên	Độ lớn (byte)
Phiên bản	4
Số lượng đầu vào	Tùy biến từ 1 đến 9
Danh sách đầu vào	Tùy biến
Số lượng đầu ra	Tùy biến từ 1 đến 9
Danh sách đầu ra	Tùy biến
Thời gian khóa	4

6.3. Cách thực thi script và loại script phổ thông

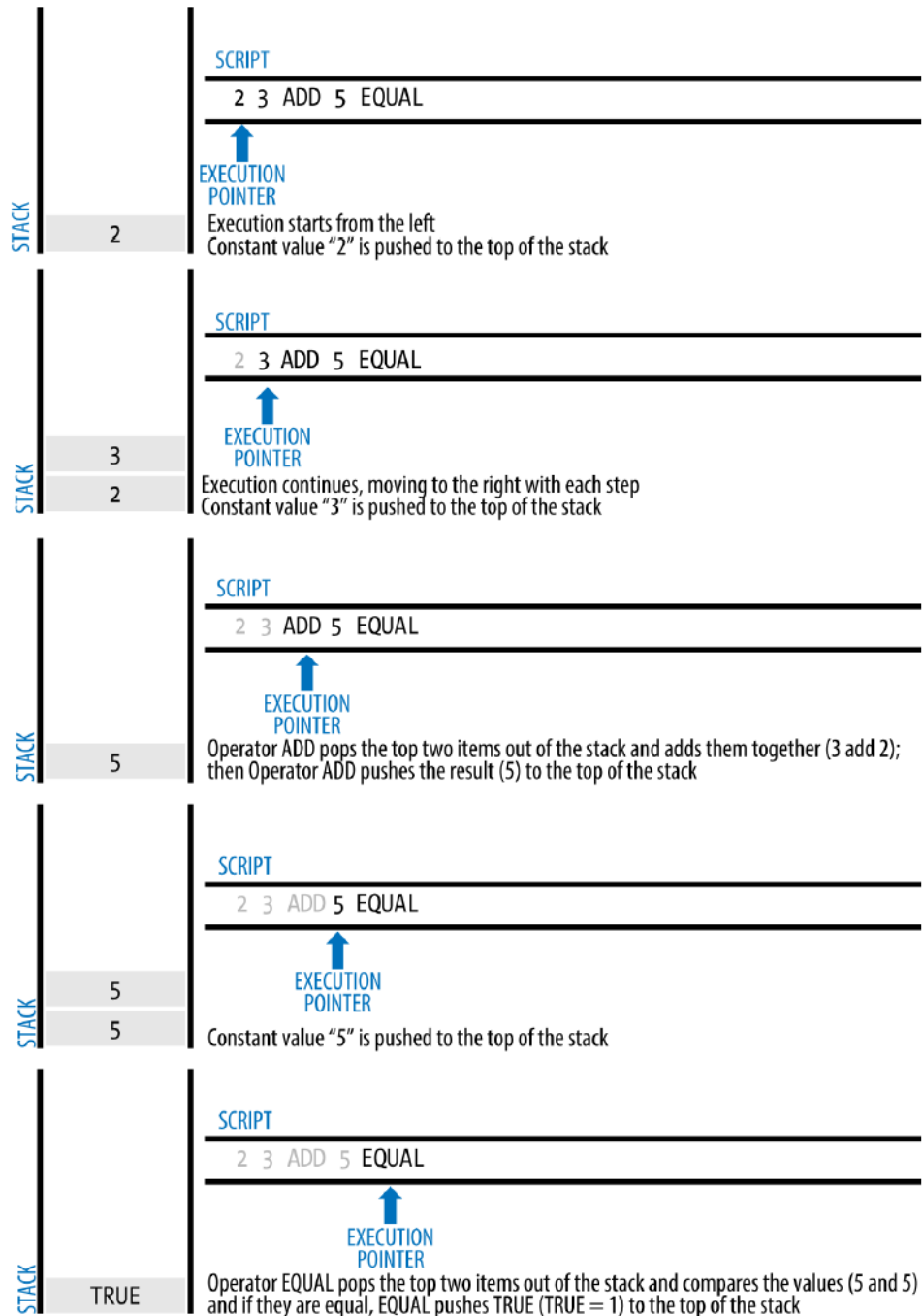
6.3.1. Script và cách thức thực thi

Trước khi đi vào tìm hiểu về loại script mặc định của bitcoin, chúng ta cần tìm hiểu cách thức các script thực thi như thế nào. Để đơn giản, hãy thử ví dụ với đoạn script sau:

¹¹ BIP-0068: Relative lock-time using consensus-enforced sequence numbers. Nguồn: https://en.bitcoin.it/wiki/BIP_0068.

2 3 OP_ADD 5 OP_EQUAL

Quá trình thực thi của đoạn script trên sẽ diễn ra như sau:



Hình 6.1: Quy trình thực thi của script [12]

Như vậy chúng ta đã biết được cách thức thực thi của script trong bitcoin. Trong giao dịch bitcoin được trình bày phía trên, có hai loại script được nhắc tới:

- Loại script đầu tiên là *locking script* (thuộc đầu ra) để tạo ra một thử thách chứng minh sự sở hữu của đầu ra giao dịch này;
- Loại script thứ hai là *unlocking script* (thuộc đầu vào) dùng để giải thử thách được đặt ra với đầu ra tương ứng mà đầu vào này sử dụng, chứng minh rằng mình có quyền sử dụng đầu ra giao dịch đó.

6.3.2. Pay-to-Public-Key-Hash script

Một phần lớn các giao dịch trên hệ thống bitcoin sử dụng loại script Pay-to-Public-Key-Hash (P2PKH) để khóa .

Nhìn lại ví dụ thanh toán ở cửa hàng tiện lợi bên trên, đầu ra gửi tiền của chúng ta cho cửa hàng được khóa bởi locking script có định dạng như sau:

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

Giá trị *pubKeyHash* ở trên sẽ tương ứng với địa chỉ của cửa hàng sau khi được giải mã Base58Check. Giá trị này sẽ được hiển thị dưới định dạng mã hóa hệ 16 thay vì Base58Check nên chúng ta sẽ không thấy được ký tự ‘1’ đứng đầu như thường lệ.

Nếu cửa hàng muốn sử dụng đầu ra này để làm đầu vào cho giao dịch của mình, cửa hàng phải cung cấp một unlocking script có định dạng như sau:

```
<sig> <pubKey>
```

Trong đó, *sig* sẽ là chữ ký điện tử được tạo ra bởi khóa bí mật của cửa hàng (quá trình này sẽ được trình bày ở phần tiếp theo), và *pubKey* sẽ là khóa công khai của cửa hàng (dạng điểm, chưa qua quá trình HASH160).

Sau khi đã có locking script và unlocking script, unlocking script sẽ được nối vào sau locking script để tạo thành một script hoàn chỉnh, và quá trình xác minh sự sở hữu được tiến hành:

```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY  
OP_CHECKSIG
```

Trong đó, các lệnh sẽ có nhiệm vụ tương ứng như sau:

- OP_DUP sẽ tạo ra một bản sao cho phần tử đầu stack và thêm vào stack;
- OP_HASH160 sẽ thực hiện phép HASH160 cho phần tử đầu stack, loại phần tử đó ra khỏi stack, sau đó thêm giá trị băm vào stack;
- OP_EQUALVERIFY sẽ thực hiện phép so sánh bằng với hai phần tử đầu stack. Nếu hai phần tử đó khác nhau, quy trình thực thi sẽ thất bại và dừng lại. Ngược lại, nếu hai phần tử đó giống nhau, quy trình sẽ tiếp tục thực thi và hai phần tử đó sẽ được loại khỏi stack;
- Cuối cùng, lệnh OP_CHECKSIG sẽ thực hiện việc kiểm tra chữ ký điện tử với hai phần tử còn lại trong stack, và loại hai phần tử đó ra khỏi stack. Nếu kiểm tra thành công, giá trị *true* sẽ được thêm vào stack, ngược lại, nếu kiểm tra không thành công thì giá trị *false* sẽ được thêm vào stack.

Quá trình thực thi thành công khi và chỉ khi trong stack chỉ còn lại đúng một giá trị *true*. Các trường hợp còn lại được coi như là không thành công và không có quyền sử dụng đầu ra tương ứng.

6.4. Chữ ký điện tử

Như có nhắc tới trong quá trình cấu tạo đầu vào giao dịch, chúng ta cần thực hiện phép ký điện tử lên toàn bộ giao dịch để có thể chứng minh rằng mình có quyền được sử dụng đầu ra tương ứng. Trong bitcoin, thuật toán chữ ký điện tử được sử dụng là thuật toán Elliptic Curve Digital Signature Algorithm (ECDSA), một thuật toán nằm trong hệ thống các thuật toán mật mã đường cong elliptic.

Chữ ký điện tử giúp giải quyết ba vấn đề trong bitcoin:

- Chữ ký điện tử giúp chứng minh một người là người sở hữu tài sản;
- Chữ ký điện tử là một bằng chứng chống thoái thác trách nhiệm;
- Chữ ký điện tử giúp chứng minh nội dung giao dịch chưa bị thay đổi dưới bất kỳ hình thức nào.

6.4.1. Thực hiện ký

Quá trình tạo ra chữ ký có thể được tóm gọn bằng công thức sau:

$$\text{Signature} = F_{\text{sign}}(F_{\text{hash}}(m), pk) \quad 6.1$$

Trong đó:

- *Signature* – là dữ ký điện tử cuối cùng,
- *m* – là dữ liệu của giao dịch cần ký,
- *F_{hash}* – là hàm băm sử dụng trước khi thực hiện ký,
- *pk* – là khóa bí mật sử dụng để ký,
- *F_{sign}* – là hàm thực hiện việc ký.

Vì chi tiết quy trình ký và nền tảng toán học của ECDSA tương đối phức tạp nên quá trình nghiên cứu em có lược bỏ qua nghiên cứu phần này.

6.4.2. Xâu chuỗi chữ ký (DER)

Distinguished Encoding Rules (DER) là một trong những tiêu chuẩn xâu chuỗi các dạng dữ liệu trừu tượng được đề ra bởi bộ ASN.1 [18]. Chúng ta sẽ xem qua chữ ký của chúng ta được mã hóa dưới định dạng DER này như thế nào trong giao dịch mua hàng mẫu ở bên trên.

Chữ ký của chúng ta là đoạn được in đậm ở giao dịch thô bên dưới.

```
010000000130d73640da0ed6ba686a50d5b7c4014eb16
8f90827c0104a96da74720a7157c1000000006b483045
022100deca2ec2e58599d99bf86528d3c13efa3b4d886
04248ca87e430e768dd4c524b022030e3a7dcfc8e0ba3
652cecee336987f87f89a1d257b25e18f68d4477eff60
8a00121026a0c583ad692cd43458154898b4ed3de261b
e6826d249af48397f171fcbe424cfffffffff02983a000
000000001976a9144538024caf2fd1dec379dee704c
cc488a53139488ac144a010000000001976a914746ca
5f027d7abb2351ef4501f85282c2ecf3c6a88ac00000000
```

Chú ý rằng byte mang giá trị 48 phía trước là độ dài của chữ ký chúng ta (72 byte).
Cấu tạo của chữ ký như sau:

- 30 – Ký hiệu bắt đầu chuỗi DER;
- 45 – Độ dài chuỗi DER (69 byte);
- 02 – Báo hiệu có một số nguyên kè sau;
- 21 – Độ dài số nguyên (33 byte);
- R – Số nguyên R (00deca...4c524b);
- 02 – Có một số nguyên kè sau;
- 20 – Độ dài số nguyên (32 byte);
- S – Số nguyên S (30e3a7...f608a0);
- 01 – Cờ hiệu cho loại chữ ký được sử dụng (không thuộc chuỗi DER).

6.4.3. Các loại chữ ký

Theo như tài liệu về giao dịch bitcoin [19], trong giao dịch, chữ ký điện tử có thể được áp dụng với nhiều thành phần khác nhau, mỗi loại được thể hiện bởi một giá trị cờ hiệu nằm cuối chuỗi DER. Các giá trị có thể được trình bày ở bảng dưới.

Bảng 6.7: Loại chữ ký và giá trị cờ hiệu tương ứng

Loại chữ ký	Giá trị
SIGHASH_ALL	0x01
SIGHASH_NONE	0x02
SIGHASH_SINGLE	0x03

Ý nghĩa của từng loại chữ ký như sau:

- SIGHASH_ALL: Đây là loại mặc định, chữ ký áp dụng cho toàn bộ đầu vào và đầu ra của giao dịch, bảo vệ mọi thứ trừ script chữ ký;
- SIGHASH_NONE: Ký toàn bộ đầu vào nhưng không cho đầu ra nào. Loại ký nào tạo điều kiện cho bất kỳ ai có khả năng thay đổi đầu ra theo mong muốn của bản thân họ (giống như séc trống);
- SIGHASH_SINGLE: Ký một đầu ra duy nhất tương ứng với đầu vào mình chọn (đầu ra và đầu vào này cần có chung số thứ tự trong giao dịch đang

ký), đảm bảo rằng mình bảo vệ phần giao dịch của mình trong khi vẫn cho người khác thực hiện việc ký của riêng họ. Tất cả các đầu vào đều được chữ ký bảo vệ.

Bên cạnh ba loại chữ ký trên, ta còn có hiệu phụ `SIGHASH_ANYONECANPAY` mang giá trị `0x80` có thể sử dụng chung với ba loại chữ ký trên (áp dụng phép OR) để tạo ra ba loại chữ ký mới:

- `SIGHASH_ALL` | `SIGHASH_ANYONECANPAY`: Ký toàn bộ đầu ra nhưng chỉ ký một đầu vào được chỉ định. Việc này khiến cho người khác có quyền thêm vào giao dịch hoặc rút khỏi giao dịch đầu vào của họ, nhưng họ không thể thay đổi điểm đến của giao dịch hoặc số tiền giao dịch;
- `SIGHASH_NONE` | `SIGHASH_ANYONECANPAY`: Chỉ ký một đầu vào tương ứng nhưng không ký đầu ra nào cả. Điều này khiến cho bất kỳ một ai đều có quyền thêm hoặc bớt đầu vào và đầu ra của giao dịch;
- `SIGHASH_SINGLE` | `SIGHASH_ANYONECANPAY`: Ký một đầu vào và một đầu ra tương ứng theo số thứ tự, tạo điều kiện cho bất kỳ ai có thể thêm hoặc bớt đầu vào và đầu ra của họ trong giao dịch.

6.4.4. Ký giao dịch bao gồm nhiều đầu vào

Một trong những khó khăn em gặp phải trong quá trình thực hiện đề tài đó là việc ký một giao dịch bao gồm nhiều đầu vào. Đa số các tài liệu em tìm hiểu đều chỉ hướng dẫn việc ký một giao dịch có một đầu vào duy nhất. Tuy nhiên, để ký một giao dịch có nhiều đầu vào thì đó lại là một vấn đề phức tạp hơn.

Để ký một giao dịch có nhiều đầu vào, ta sẽ áp dụng quy trình ký tương tự như quy trình ký một giao dịch có một đầu vào một cách riêng biệt cho từng đầu vào của giao dịch đó. Tuy nhiên, trước khi ta thực sự thực hiện việc ký cho đầu vào hiện tại, ta phải tạm thời loại bỏ toàn bộ unlocking script của các đầu vào còn lại đi. Trong khi đó, unlocking script của đầu vào hiện tại sẽ là locking script của UTXO mà đầu vào đó được tham chiếu tới.

Lý do của việc này sẽ là để loại bỏ việc chữ ký của những đầu vào đã được ký bị ảnh hưởng bởi unlocking script của một trong những đầu vào tiếp theo bị thay đổi khi ký các đầu vào tiếp theo đó, dẫn tới việc xác thực chữ ký không thành công. Nếu chúng ta loại bỏ hết các unlocking script của các đầu vào còn lại, thì chữ ký điện tử của đầu vào nào sẽ không bị phụ thuộc bởi các đầu vào còn lại.

6.5. Một số thông tin thêm về giao dịch

6.5.1. Phí giao dịch

Nếu để ý giao dịch mua hàng bên trên, ta có thể thấy tổng số tiền đầu ra trong giao dịch có tổng không bằng tổng giá trị của đầu vào giao dịch. Sự chênh lệch này được gọi là phí giao dịch. Lượng phí thấp nhất mà một giao dịch cần phải có là 1 satoshi cho một byte của giao dịch sau khi xâu chuỗi hóa.

Lượng phí giao dịch này sẽ được lấy bởi người đào thành công block mà giao dịch này thuộc về được trình bày ở Chương 2. Vì lượng phí này một phần quyết định thu nhập của người đào nên thông thường các giao dịch có lượng phí cao hơn thường sẽ được ưu tiên nhiều hơn, dẫn tới việc những giao dịch có mức phí thấp thường phải đợi lâu hoặc có thể là không bao giờ được đào.

Để giải quyết vấn đề này, ta có hai cách làm:

- Rút giao dịch cũ và tạo một giao dịch mới với phí cao hơn;
- Sử dụng tính năng Opt-in Replace-by-Fee để điều chỉnh phí giao dịch, tính năng này lần đầu được công bố vào phiên bản Bitcoin Core 0.12 [20].

6.5.2. Nhúng nội dung vào giao dịch

Để nhúng một nội dung không thể thay đổi vào giao dịch thì ta có thể sử dụng đầu ra nội dung. Một đầu ra nội dung là một đầu ra không có satoshi, và locking scrip có dạng:

```
OP_RETURN <data>
```

Với dạng đầu ra này, hệ thống khi gặp locking script này sẽ thực hiện trả về lượng dữ liệu phía sau lệnh OP_RETURN.

Tuy đây là một cách dùng để đính kèm nội dung giao dịch, nội dung này sẽ được lưu trên blockchain, tốn một dung lượng nhất định và sẽ khiến cho phí giao dịch bị tăng lên. Vì lý do này mà việc đính kèm nội dung giao dịch như này không được các nhà phát triển Bitcoin khuyến khích sử dụng.

6.5.3. Phát giao dịch lên mạng lưới bitcoin

Để phát giao dịch lên mạng lưới bitcoin, chúng ta có thể dùng hàm được cung cấp sẵn của Bitcoin Core là `sendrawtransaction`, tiếp nối bởi xâu giao dịch thô để gửi lên mạng lưới bitcoin. Nếu gửi thành công, hàm sẽ trả về TXID tương ứng của giao dịch.

Chương 7

Một số tính năng nhỏ và giao diện tương tác trên dòng lệnh

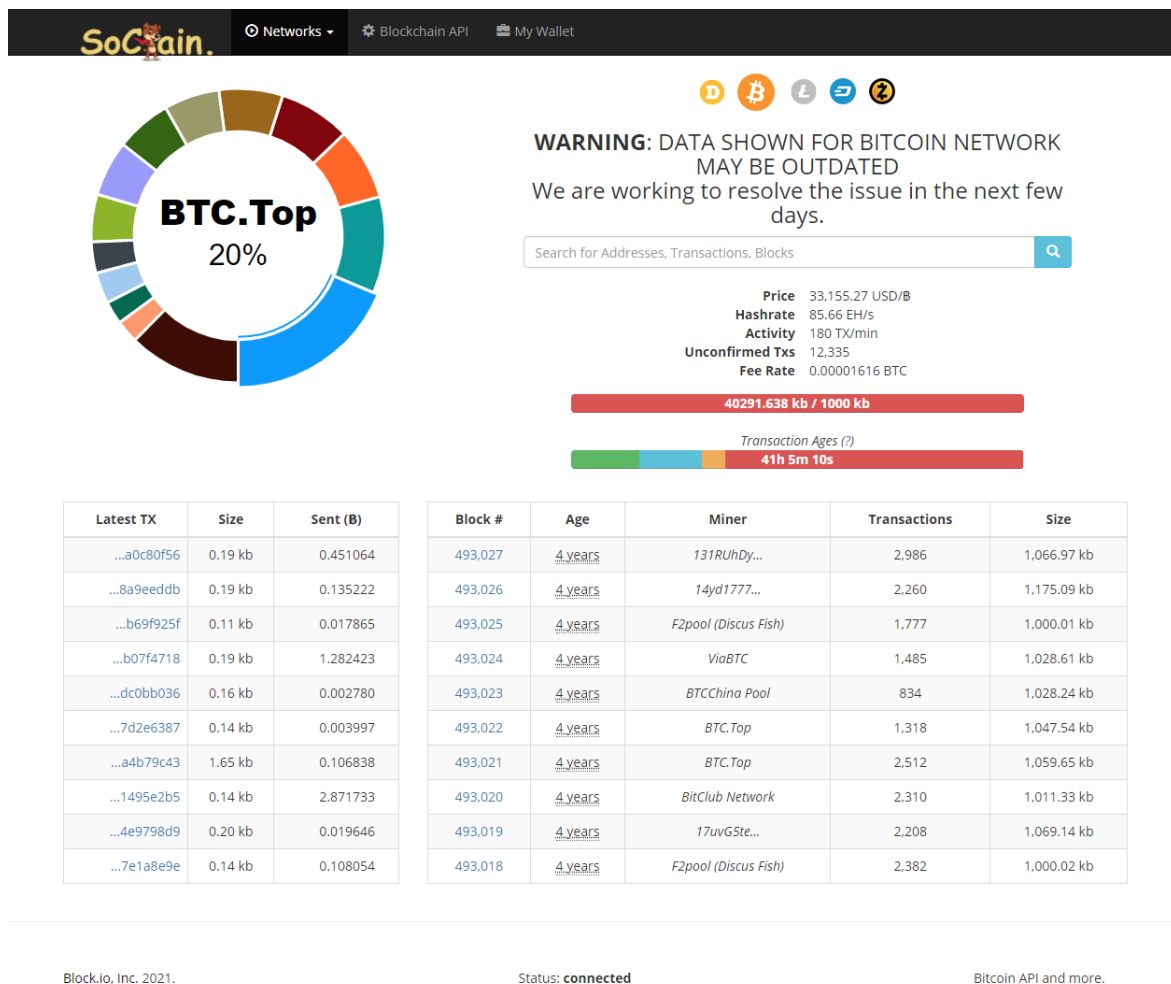
Các chương trước đã trình bày những kiến thức nền tảng về bitcoin và Bitcoin Core. Chương này sẽ đi vào những tính năng hỗ trợ người dùng và quá trình cài đặt giao diện tương tác dòng lệnh của ví.

7.1. Các tính năng hỗ trợ

Để tiện lợi cho người dùng, một ví điện tử cần có ít nhất hai tính năng là vấn tin số dư tài khoản và truy vấn lịch sử giao dịch. Hai tính năng này cần phải thực hiện các truy vấn trên blockchain. Tuy nhiên, Bitcoin Core không có hàm nào hỗ trợ hai tính năng này. Bên cạnh đó, nếu em thực hiện truy vấn thông qua Bitcoin Core, người dùng cần phải tải toàn bộ blockchain về, điều này không thực sự hợp lý. Vì các lý do trên, em cần phải tìm một công cụ thay thế giúp thực hiện những công việc này.

7.1.1. Chain.so – API hỗ trợ truy vấn trên blockchain

Chain.so, cũng như blockchain.com được giới thiệu bên trên, là một trình khám phá blockchain. Ngoài là trình khám phá blockchain, chain.so còn cung cấp một tập API miễn phí thông qua giao thức HTTP giúp thực hiện các thao tác tương ứng.



Hình 7.1: Giao diện chính của trang chain.so

Hiện tại, các công cụ truy vấn blockchain là tương đối nhiều. Một số trang cung cấp công cụ truy vấn bên cạnh chain.so có thể kể qua là: *blockchain.com*, *blockstream.info*, hay *blockchair.com*, ...

Lý do sự lựa chọn không có gì ngoài giá trị trả về của dịch vụ này phù hợp với nhu cầu của đề tài và dễ dàng sử dụng. Thực sự tất cả các sự lựa chọn em đã xem qua đều tốt, và nếu để so sánh sự hơn kém giữa các dịch vụ thì khá khó khăn.

Thư viện cURL và jsoncpp

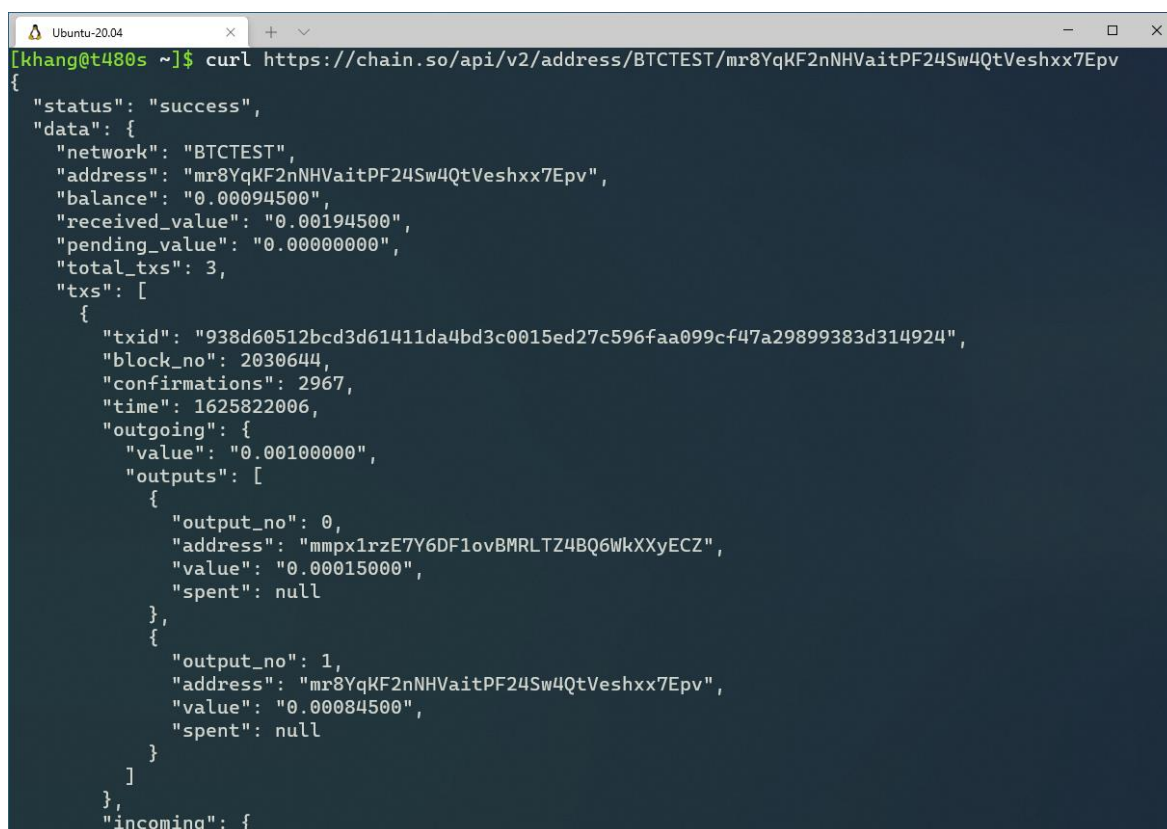
Để thực hiện các giao thức HTTP gọi API, chương trình của em cần hai thư viện hỗ trợ là cURL và jsoncpp. Trong đó, cURL là thư viện giúp thực hiện việc cài đặt một HTTP client, còn jsoncpp là một thư viện hỗ trợ xử lý JSON cho C++, giúp

chương trình có thể đọc và xử lý những đối tượng JSON được trả về bởi API một cách dễ dàng.

7.1.2. Một số tính năng hỗ trợ

Chương trình của em có hỗ trợ hai tính năng cơ bản là vấn tin số dư và truy vấn lịch sử giao dịch. Với chain.so, hai địa chỉ cần truy vấn để thực hiện hai công việc trên cùng với một địa chỉ:

<https://chain.so/api/v2/address/{network}/{address}>



```
Ubuntu-20.04
[khang@t480s ~]$ curl https://chain.so/api/v2/address/BTCTEST/mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epv
{
  "status": "success",
  "data": {
    "network": "BTCTEST",
    "address": "mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epv",
    "balance": "0.00094500",
    "received_value": "0.00194500",
    "pending_value": "0.00000000",
    "total_txs": 3,
    "txs": [
      {
        "txid": "938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a29899383d314924",
        "block_no": 2030644,
        "confirmations": 2967,
        "time": 1625822006,
        "outgoing": {
          "value": "0.00100000",
          "outputs": [
            {
              "output_no": 0,
              "address": "mmpx1rzE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ",
              "value": "0.00015000",
              "spent": null
            },
            {
              "output_no": 1,
              "address": "mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epv",
              "value": "0.00084500",
              "spent": null
            }
          ]
        },
        "incoming": {
```

Hình 7.2: Ví dụ thực hiện lời gọi lấy thông tin địa chỉ trên testnet

Tính năng tiếp theo em cần sử dụng đến chain.so là truy vấn thông tin giao dịch. Để truy vấn thông tin giao dịch, ta có thể sử dụng địa chỉ:

https://chain.so/api/v2/get_tx/{network}/{txid}

```
[khang@t480s ~]$ curl https://chain.so/api/v2/get_tx/BCTEST/938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a29899383d314924
{
  "status": "success",
  "data": {
    "network": "BCTEST",
    "txid": "938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a29899383d314924",
    "blockhash": "00000000000000840ecc5d70cc100037960e18338189dc5edc6b9e39bf3ada67",
    "confirmations": 2969,
    "time": 1625822058,
    "inputs": [
      {
        "input_no": 0,
        "value": "0.00100000",
        "address": "mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epv",
        "type": "pubkeyhash",
        "script": "3045022100deca2ec2e58599d99bf86520d3c13efa3b4d80604248ca07e430e768dd4c524b022030e3a7dcfc8e0ba3652cecee336987f87f89a1d257b25e18f68d4477efff608a001 026a0c583ad692cd4345815d898b4ed3de261be6826d249af40397f171fcbe424c",
        "witness": null,
        "from_output": {
          "txid": "c157710a7274da964a10c02708f968b14e01c4b7d5506a68bad60eda4036d730",
          "output_no": 0
        }
      }
    ],
    "outputs": [
      {
        "output_no": 0,
        "value": "0.00015000",
        "address": "mmpx1rzE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ",

```

Hình 7.3: Ví dụ lời gọi lấy thông tin giao dịch

Ngoài ra, để giúp người dùng tạo giao dịch dễ dàng, em còn sử dụng thêm chức năng truy vấn UTXO của chain.so sử dụng đường dẫn:

https://chain.so/api/v2/get_tx_unspent/{network}/{address}

```
[khang@t480s ~]$ curl https://chain.so/api/v2/get_tx_unspent/BCTEST/mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epv
{
  "status": "success",
  "data": {
    "network": "BCTEST",
    "address": "mr8YqKF2nNHVaitPF24Sw4QtVeshxx7Epv",
    "txs": [
      {
        "txid": "58e2afe86c6904734d2216275914cc9741a84840faeef3b3a96f5b38d004ae2d",
        "output_no": 0,
        "script_asm": "OP_DUP OP_HASH160 746ca5f027d7abb2351ef4501f85282c2ecf3c6a OP_EQUALVERIFY OP_CHECKSIG",
        "script_hex": "76a914746ca5f027d7abb2351ef4501f85282c2ecf3c6a88ac",
        "value": "0.00010000",
        "confirmations": 26175,
        "time": 1625461086
      },
      {
        "txid": "938d60512bcd3d61411da4bd3c0015ed27c596faa099cf47a29899383d314924",
        "output_no": 1,
        "script_asm": "OP_DUP OP_HASH160 746ca5f027d7abb2351ef4501f85282c2ecf3c6a OP_EQUALVERIFY OP_CHECKSIG",
        "script_hex": "76a914746ca5f027d7abb2351ef4501f85282c2ecf3c6a88ac",
        "value": "0.00084500",
        "confirmations": 2970,
        "time": 1625822058
      }
    ]
  }
}
```

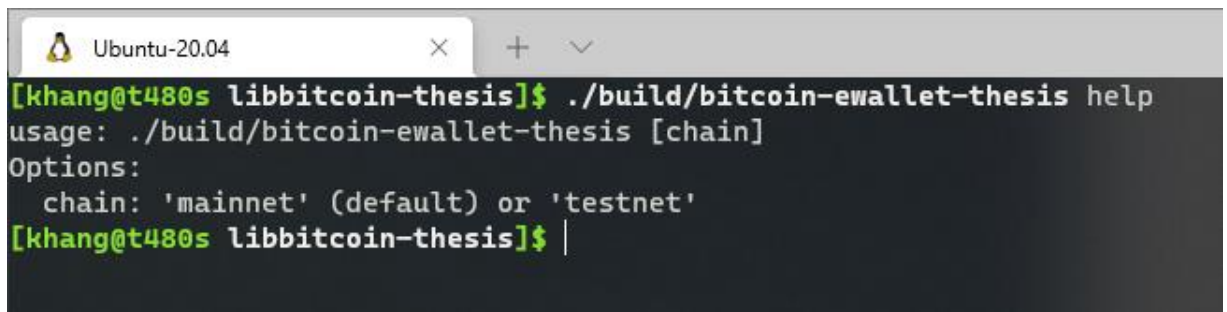
Hình 7.4: Ví dụ lời gọi lấy thông tin UTXO của địa chỉ

7.2. Giao diện tương tác trên dòng lệnh

Một trong những lý do chương trình được thiết kế chạy trên giao diện tương tác dòng lệnh thay vì giao diện đồ họa đầy đủ là giới hạn thời gian. Thứ hai là do em chưa có kinh nghiệm về lập trình giao diện đồ họa sử dụng C++. Trong thời gian xây dựng nền tảng ví, em cũng có thử nghiệm với bộ thư viện Qt nhưng không có kết quả khả quan, sau đó em đã quyết định sử dụng tương tác dòng lệnh làm giao diện của ứng dụng.

7.2.1. Giao diện chính

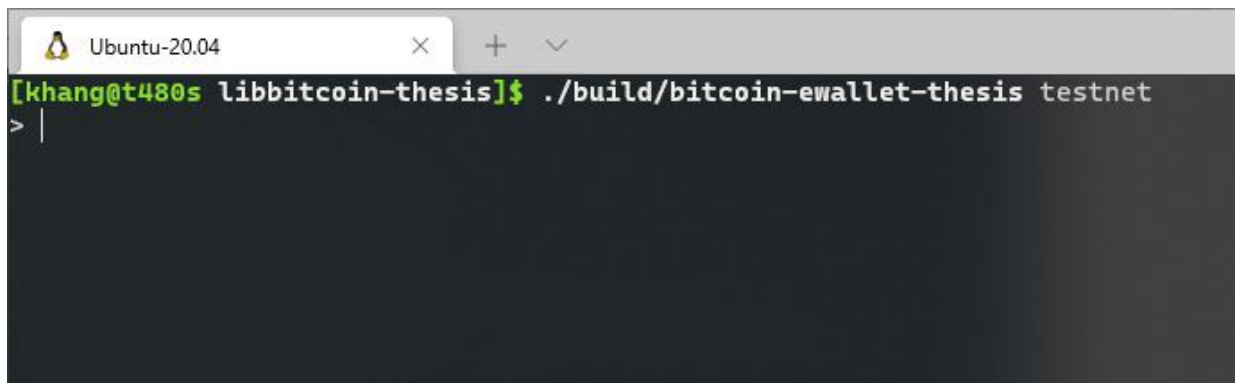
Để khởi động chương trình, ta chỉ cần thực thi ứng dụng sau khi đã biên dịch thành công. Mặc định, chương trình sẽ khởi động và sử dụng mainnet. Việc thay đổi mạng lưới sử dụng rất đơn giản, trong quá trình khởi động ứng dụng, ta chỉ cần truyền thêm một tham số dòng lệnh để chương trình biết mình cần sử dụng testnet.



```
Ubuntu-20.04
[khang@t480s libbitcoin-thesis]$ ./build/bitcoin-ewallet-thesis help
usage: ./build/bitcoin-ewallet-thesis [chain]
Options:
  chain: 'mainnet' (default) or 'testnet'
[khang@t480s libbitcoin-thesis]$ |
```

Hình 7.5: Hướng dẫn khởi động chương trình

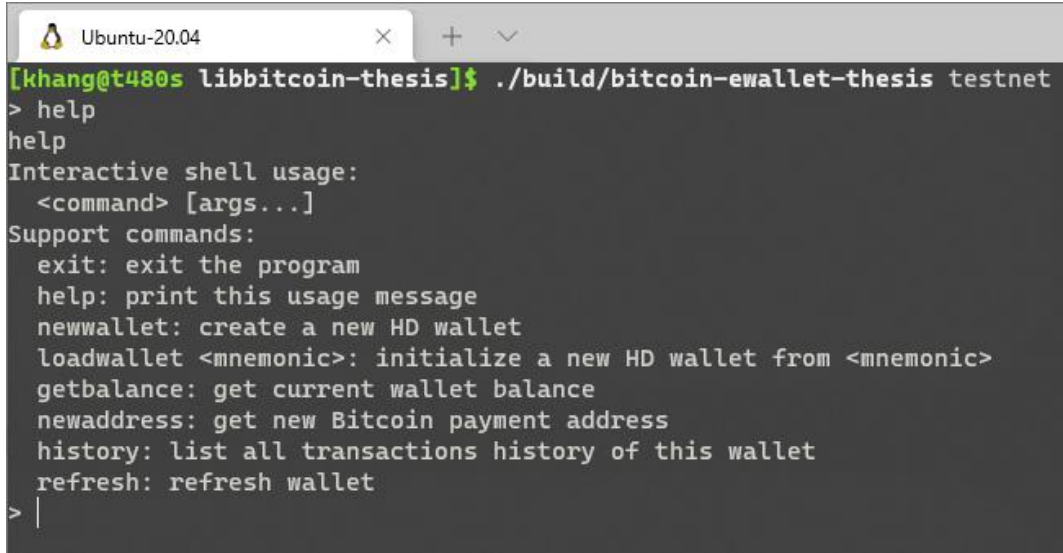
Sau khi khởi động thành công, giao diện ví ban đầu sẽ như sau:



```
Ubuntu-20.04
[khang@t480s libbitcoin-thesis]$ ./build/bitcoin-ewallet-thesis testnet
> |
```

Hình 7.6: Giao diện tương tác dòng lệnh

Để biết thêm thông tin về các lệnh được hỗ trợ, chúng ta có thể chạy lệnh *help*:

A screenshot of a terminal window titled 'Ubuntu-20.04'. The prompt is '[khang@t480s libbitcoin-thesis]\$'. The user has entered './build/bitcoin-ewallet-thesis testnet' and then '> help'. The output shows 'Interactive shell usage: <command> [args...]' followed by 'Support commands:' and a list of commands: 'exit: exit the program', 'help: print this usage message', 'newwallet: create a new HD wallet', 'loadwallet <mnemonic>: initialize a new HD wallet from <mnemonic>', 'getbalance: get current wallet balance', 'newaddress: get new Bitcoin payment address', 'history: list all transactions history of this wallet', and 'refresh: refresh wallet'. The prompt '> |' is visible at the bottom.

```
[khang@t480s libbitcoin-thesis]$ ./build/bitcoin-ewallet-thesis testnet
> help
help
Interactive shell usage:
  <command> [args...]
Support commands:
  exit: exit the program
  help: print this usage message
  newwallet: create a new HD wallet
  loadwallet <mnemonic>: initialize a new HD wallet from <mnemonic>
  getbalance: get current wallet balance
  newaddress: get new Bitcoin payment address
  history: list all transactions history of this wallet
  refresh: refresh wallet
> |
```

Hình 7.7: Các lệnh chương trình hỗ trợ

7.2.2. Tạo giao dịch

Để tạo giao dịch, ta cần phải khởi tạo một ví sử dụng lệnh *newwallet* hoặc tải lại một ví bằng lệnh *loadwallet* nối theo sau bởi chuỗi mnemonic. Sau khi có được ví, ta có thể sử dụng lệnh *newtransaction* để vào dòng lệnh tạo giao dịch. Trong chế độ này, tập lệnh hỗ trợ sẽ được thay đổi và trước dấu '>' tại nơi nhập lệnh sẽ có thêm một số ký tự giúp cho người dùng phân biệt rằng mình đang ở trong trạng thái nào.


```
Ubuntu-20.04 [khang@t480s libbitcoin-thesis]$ ./build/bitcoin-ewallet-thesis testnet
> loadwallet bench spice side slot orange brother one fetch sting evil prison
good seminar ring because gasp shine hour holiday swap ripple dust build vib
rant
Passphrase (leave empty if none):
Wallet successfully created
Your mnemonic is:
bench spice side slot orange brother one fetch sting evil prison good seminar
ring because gasp shine hour holiday swap ripple dust build vibrant
Exploring wallet... success
Wallet initialized successfully
Your wallet balance:
  Balance: 0.00185262 BTC
  Pending: 0 BTC
  Equals to: 1,357,401.269324 VND
> newtransaction
[t] > help
Support transaction building commands:
  exit: cancel transaction
  help: print this usage message
  done: confirm transaction
  addinput: add an input to transaction
  addoutput: add an output to transaction
  addmessage: add a note to transaction
  refresh: refresh wallet

Note: you do not need to specify the change output,
it will automatically added when you finish the transaction
(use the 'done' command)
[t] > |
```

Hình 7.8: Quy trình vào chế độ tạo giao dịch

Để thêm đầu vào và đầu ra giao dịch, chúng ta chỉ cần sử dụng lệnh *addinput* và *addoutput* tương ứng và làm theo hướng dẫn của chương trình. Sau khi hoàn thành giao dịch, ta sẽ sử dụng lệnh *done* để chọn phí. Nếu chúng ta muốn hủy, ta sử dụng lệnh *exit* để hủy quá trình cấu tạo giao dịch.

```
Ubuntu-20.04
address: add a note to transaction
refresh: refresh wallet

Note: you do not need to specify the change output,
it will automatically added when you finish the transaction
(use the 'done' command)
[t] > addinput
Unspent transactions:
(1): TXID: ac68359c03455b3a258cf61177e4f67a321a00f91b764eb53283917ed20cbf83 | Value: 0.00001 BTC
(2): TXID: 1dbd97025b0f6dc5fd6f4fd263fde70e8fcdaf46eb1271a331c4080560893546 | Value: 0.001 BTC
(3): TXID: d0cb1d8168f6c4cc97c27b150fbd4edbef2f5e2f80a6e146bf80d8c03ca2e191 | Value: 0.00037358 BTC
(4): TXID: ac68359c03455b3a258cf61177e4f67a321a00f91b764eb53283917ed20cbf83 | Value: 0.00046904 BTC
Choose a transaction to use as input (0 to cancel): 1
Picked transaction #1
You can spend up to: 0.00001 BTC
[t] > addoutput
Enter receive address: mmpx1rzE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ
Left over BTC to spend: 0.00001 BTC
Enter amount to send (in BTC): 0.000001
[t] > done
Fastest fee (per byte) is 102
Your maximum fee (per byte) is 3
Please enter your transaction fee (per byte): 2
Your transaction fee is: 452 satoshis
Making change output... done
Signing transaction... done
Your raw transaction: 01000000183bf0cd27e918332b54e761bf9001a327af6e47711f68c253a5b45039c3568ac000000
006a473044022043c033e5f1be22bfa78acb31f189d94ec5d6bdeb80a85271065fdf22ee768d802200ef5d49b608e727353b2
6a3a90299eed5e6270e78dcb196876765bb99735fc940121034b846cb4e084773aefb082102d5375b01f9d379fc9f7ba38a02c
ac8eae0d1101fffffffff026400000000000000001976a9144538024caf2fd1dec379dee704ccc488a53139488acc00100000000
00001976a91467e15df5c472c1e72b94a002c6ea28571c0b576688ac00000000
Would you like to broadcast this transaction? (Y/n) |
```

Hình 7.9: Ví dụ quá trình tạo giao dịch

Sau khi hoàn thành giao dịch, người dùng được lựa chọn có muốn phát giao dịch lên mạng lưới bitcoin hay không. Nếu người dùng lựa chọn có (bằng cách nhập ‘y’, hoặc ‘Y’, hoặc không nhập gì cả), giao dịch sẽ được tiến hành phát lên mạng lưới.

7.2.3. Hạn chế và thiếu sót

Bởi vì chương trình được xây dựng xoay quanh hệ điều hành Linux và sử dụng dòng lệnh làm giao diện chính, người dùng cần một số kiến thức cơ bản về dòng lệnh để có thể sử dụng ứng dụng. Ngoài ra, một ví điện tử với giao diện dòng lệnh không phải là một trong những ứng dụng tốt nhất hiện tại.

Trong quá trình khảo sát những ví điện tử hỗ trợ bitcoin hiện nay, có rất nhiều ví điện tử có giao diện rất đẹp và những chức năng rất nổi bật như có khả năng chuyển đổi từ đồng tiền ảo này sang một đồng tiền ảo khác ngay trong ứng dụng (ứng dụng ví có tích hợp sàn giao dịch). Một số ví điện tử có hỗ trợ lên tới hơn 1700 loại tiền ảo khác nhau, đơn cử như ví điện tử Coinomi. Một số ví điện tử khác còn hỗ trợ cho các

ứng dụng phân tán (decentralized apps – dApps) như Trust Wallet, hay kể cả hỗ trợ tài sản dưới dạng *non-fungible token (NFT)*.

Bên cạnh các ví điện tử, ta còn có các *ví phần cứng (hardware wallet)*, là một loại ví được thiết kế sử dụng như một thiết bị riêng biệt, không cần kết nối mạng. Điều này giảm thiểu nguy cơ bị tin tặc tấn công qua mạng rất nhiều.

Một trong những tính năng bảo mật khác mà một số ví điện tử có hỗ trợ là khả năng sử dụng chung với một ví phần cứng. Một ví dụ có thể kể đến là ví điện tử Exodus và ví phần cứng Trezor [21].

Chương 8

Thử nghiệm chương trình

Việc thử nghiệm cho chương trình là một việc không thể thiếu, đặc biệt là với một ứng dụng liên quan tới tài sản cá nhân. Chương này được dùng để trình bày ngắn gọn những thử nghiệm mà em đã thực hiện cho chương trình của em.

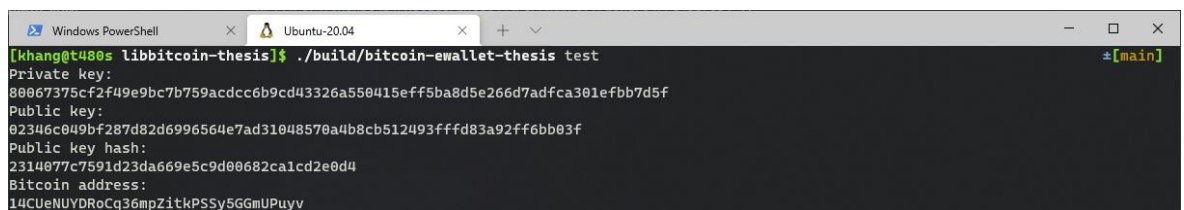
8.1. Thử nghiệm lý thuyết

Trong quá trình cài đặt cấu trúc nền tảng của ví HD và cấu tạo giao dịch, em có thử nghiệm chương trình của em cài đặt (hạn chế tối đa việc sử dụng cấu trúc được thư viện cung cấp) với một số bộ kiểm thử có sẵn. Với những tính năng không có bộ kiểm thử, em sẽ thử nghiệm và xác minh tính đúng đắn bằng cách so sánh đầu ra giữa cách cài đặt của em và đầu ra của các cấu trúc tương ứng được thư viện cung cấp. Các thử nghiệm này cũng được đính kèm trong mã nguồn của chương trình.

Một số thử nghiệm lý thuyết em đã thử nghiệm có thể kể qua như:

- Sinh khóa và tạo địa chỉ bitcoin;
- Sinh ví HD;
- Thử nghiệm quá trình dẫn xuất khóa với các test vector của BIP-0032;
- Thử nghiệm cấu tạo một giao dịch.

Để sử dụng chế độ kiểm thử, chương trình cần phải được biên dịch sử dụng chế độ Debug để có thể sử dụng một số tính năng dành riêng cho việc kiểm thử. Dưới đây là một số hình ảnh cho quá trình thử nghiệm lý thuyết của chương trình.



```
Windows PowerShell x Ubuntu-20.04 x + -
[khang@t480s libbitcoin-thesis]$ ./build/bitcoin-ewallet-thesis test
Private key:
80067375cf2f49e9bc7b759acdcc6b9cd43326a550415eff5ba8d5e266d7adfcc301efbb7d5f
Public key:
02346c049bf287d82d6996564e7ad31048570a4b8cb512493fffd83a92ff6bb03f
Public key hash:
2314077c7591d23da669e5c9d00682ca1cd2e0d4
Bitcoin address:
14CueNUYDRoCq36mpZitkPSSy5GGmUPuyv
```

Hình 8.1: Thử nghiệm quy trình tạo khóa và địa chỉ bitcoin

```

Random HD wallet generation:
Entropy: 47b829297aca7bd083a628490f388bd6bc0f264a71b821aef8bbefeb1b26fe22
Mnemonic: element scout engage void police tribe attend shallow empower keep carry pumpkin rose junior myth toddler link student title
law ticket nature weasel final
Passphrase: (none)
Master private key: tprv8ZgxMBicQKsPeKkCGi32DzKYzVYk28nFmuuZKn6kGC6YVZFGbQwYtZjYVaw2fuPDzP4wLVaDxNddRD7HqCUfbqMdeQkvUMfb8yfw3q7k6HE
Master public key: tpubD6NzVbkrYhZ4XnmzAMhcdPyfZX4gBTyAMDWLCJ93gTtwL3W3Dom8e4MQfJMGpYBodiNesRBwwjSnQDPMBQR23kBryL4kbRwvQULb1ZyHhHT
=====
Testing with Mastering bitcoin book's examples...
Test #1 passed
Test #2 passed
Test #3 passed

Testing wallet master keys generation...
Passed...

```

Hình 8.2: Thử nghiệm quy trình tạo ví HD và master key

```

Testing with BIP32 tests...
Test vector #1...
Chain m... ok
Chain m/0H... ok
Chain m/0H/1... ok
Chain m/0H/1/2H... ok
Chain m/0H/2H/2... ok
Chain m/0H/2/1000000000... ok
passed!

Test vector #2...
Chain m... ok
Chain m/0... ok
Chain m/0/2147483647H... ok
Chain m/0/2147483647H/1... ok
Chain m/0/2147483647H/1/2147483646H... ok
Chain m/0/2147483647H/1/2147483646H/2... ok
passed!

Test vector #3...
Chain m... ok
Chain m/0H... ok
passed!

All BIP32 tests passed

```

Hình 8.3: Thử nghiệm quy trình dẫn xuất xóa với các bộ kiểm thử của BIP-0032

```

Testing transaction creation process with libbitcoin's example...
ok
Raw transaction:
0100000001e9c55a9aff2d62663f24157a85d204a0ee6008f4cdd913bc916e84fc1e605ca000000006b483045022100fa510547f5906c488d946a7e0cfe9de3e99e41
51ffe8ba17edd5bc916dd5667502205628c35a06efde5ed6ed876cefc671ba7a5392a931d48bdf6418525ea0b382760121029d4f145f18762a2397ceac361033b63904
489f90e2a0b69882c9c2d0017bdd0afffffff016003b807000000001976a91442b9b7745ec8b14788f0ca7ac28150782351d44788ac00000000
passed!

Address 1: myse9rgAtArhkPoEgU71g6VPWF5jzqGr3X
Address 2: mmpxlrzE7Y6DFlovBMRLTZ4BQ6WkXyECZ
Address 3: mFzUeGZCAyS9qugJosefyJ1xd8kmSZXz5C
Raw transaction: 0100000004ec3c29fb4f45ecf5127ccd9ef41d78ee8d284d445c275e27685cc06db5eae670000000006b4830450221009fadd618e4ea52fc18d1f
d7b0bdcbe4627acf07f7ad5cfe01fc6c80d4649fc5102200a0d0692efc047ad9aa24b25484b8de30a6d0984924488e72e1c2476f10c49cf0121024673664861f87d65a
91570607d1bd8a1f0e6b83b98359b882c39f67175119c5afffffffec3c29fb4f45ecf5127ccd9ef41d78ee8d284d445c275e27685cc06db5eae670010000006a47304
402205ee758e1ec0cd69277a867cd86567ae7c12c8e87769c858392c987a1db77b25e02205073c5e3ab99b2f2887645a11c8625dd71b58e3c6980312b277d61db15149
6be012103462b7b21234ba16532797f57a3baa1d90f9c2216c4c8bc9cc4abf52853444a0ffffffffffe558df0b236b65e004053ee466737b35353626ca9472e257e32e67
1dc5bf6c43f0000000006b483045022100da0411500477fc5a2bc3349d796ce4e9bf01477e2ae0e782f45a3290b2de2db902204e91ac37168e8e0bbbaec407e088e33db
5a3a5e6ab83107a8dde13adeaad8b30121024673664861f87d65a91570607d1bd8a1f0e6b83b98359b882c39f67175119c5afffffff7720a4546326589d526c6c329
687bd36a3af2fa046d3fa9768bf9f231250b02a0100000006a473044022053f6b4a939480fb403d6070f9673ac5f4eca361e1ca1fc4968c478d8e981b65002204c64e2
478615b7904e036d29fd6896f75f8d876646534bbf75d0bd43a2f1a6f01210302f20896570c03df2a4634c27513f7a410023a4a705de47796c43818a3b8d7cffffffffff
f0410270000000000001976a9144538024caf2fd1dec379dee704ccc488a53139488ac10270000000000001976a9140534908cd49245bf9f3b65688ae8104fdaf1bdc
988ac00000000000000206a1e4b686f61206c75616e20746f74206e6768696570204649542d48434d5553e640020000000001976a914c95bdb0e9dc09cc64aa3583
beaafd4a7dc98f19f88ac00000000
Transaction size: 743 byte(s)

```

Hình 8.4: Thử nghiệm quy trình cấu tạo giao dịch

8.2. Thử nghiệm trên mạng testnet bitcoin

Với Bitcoin Core, em có thể sử dụng chế độ regtest để thực hiện các thử nghiệm về giao dịch. Tuy nhiên, với mong muốn có một môi trường giống với môi trường

mainnet nhất, em quyết định sử dụng testnet để thử nghiệm các tính năng liên quan đến giao dịch. Để có coin thử nghiệm, em sử dụng các “vòi” coin có sẵn trên mạng, đơn cử như trang: <https://testnet-faucet.mempool.co/>.

Tính năng lớn nhất em từng thử nghiệm trên testnet là giao dịch bitcoin. Em đã thử nghiệm tạo giao dịch từ một đầu vào và một đầu ra, cho đến một giao dịch có nhiều đầu vào và nhiều đầu ra cùng với đính kèm một mẫu tin trên giao dịch.

Explorer > Bitcoin Testnet > Transaction

Search your transaction, an address or a block USD

Summary

This transaction was first broadcast to the Bitcoin network on June 18, 2021 at 12:07 AM GMT+7. The transaction currently has 28,455 confirmations on the network. At the time of this transaction, 0.00030000 BTC was sent with a value of \$11.53. The current value of this transaction is now \$9.98. Learn more about [how transactions work](#).

Hash	Value	Address	Value	Address
7cda81957528be34be79b98bcf1c58bf8fc515e42034acb3c20c...	0.00000010 BTC	mmpx1rzE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ	0.00010000 BTC	myse9rgAtArhkPoEgU71g6VPWF5jqGr3X
	0.00000010 BTC	myse9rgAtArhkPoEgU71g6VPWF5jqGr3X	0.00010000 BTC	mmpx1rzE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ
	0.00000010 BTC	mmpx1rzE7Y6DF1ovBMRLTZ4BQ6WkXXyECZ	0.00010000 BTC	mfzUeGZCAyS9qugJosefyJ1xd8kmSZXz5C
	0.00000010 BTC	mfzUeGZCAyS9qugJosefyJ1xd8kmSZXz5C	0.00000000 BTC	OP_RETURN

Fee: 0.00170020 BTC (228.829 sat/B - 57.207 sat/WU - 743 bytes) 0.00030000 BTC

Hình 8.5: Thử nghiệm tạo giao dịch có nhiều đầu vào và đầu ra thành công

(Nguồn: [https://www.blockchain.com/btc-](https://www.blockchain.com/btc-testnet/tx/7cda81957528be34be79b98bcf1c58bf8fc515e42034acb3c20c13a52830b4)

testnet/tx/7cda81957528be34be79b98bcf1c58bf8fc515e42034acb3c20c13a52830b4
13)

Chương 9

Kết luận và định hướng phát triển

Chương cuối cùng này được dùng để trình bày một cách tóm lược những gì mà khóa luận đã tìm hiểu và các kết quả đạt được trong quá trình thực hiện. Qua đó có thể rút ra những nhận xét về những hướng phát triển tiềm năng trong tương lai.

9.1. Kết luận

Nhờ đề tài này, em đã tiếp nhận thêm được một lượng kiến thức rất lớn về công nghệ blockchain và cách thức hoạt động của bitcoin, với trọng tâm nằm ở ví HD và giao dịch bitcoin. Qua khóa luận, em đã tìm hiểu được:

- Cách thức hoạt động của blockchain và bitcoin;
- Cách thức tạo khóa, tạo địa chỉ bitcoin;
- Nắm được những công nghệ ví cho bitcoin, biết được những điểm mạnh và điểm yếu của từng loại công nghệ;
- Biết thêm về mã mnemonic được sử dụng để sao lưu ví;
- Hiểu được cấu trúc và các thành phần của một giao dịch bitcoin.

Thông qua những kiến thức đã tìm hiểu được, em đã có thể tạo nên được một ví điện tử cơ bản với đa số các thành phần là tự bản thân viết, không sử dụng thư viện:

- Tạo khóa và tạo địa chỉ bitcoin;
- Xây dựng ví HD;
- Xây dựng một giao dịch hoàn chỉnh, có thể đưa lên mạng lưới bitcoin.

9.2. Định hướng phát triển

Với sự phát triển ngày càng nhanh của tiền mã hóa và blockchain, đề tài này thực sự có rất nhiều hướng phát triển tiềm năng. Tuy nhiên, hướng phát triển nên được ưu

tiên hiện tại chính là xây dựng hoàn chỉnh một giao diện đồ họa, giúp người dùng dễ dàng tương tác và sử dụng ví hơn.

Tiếp đến, đề tài có thể hướng tới mở rộng hỗ trợ cho những loại tiền mã hóa phổ biến như Ethereum, Solana, Monero, ... Sau đó, đề tài cũng có khả năng mở rộng để hỗ trợ các loại SIGHASH, hay cũng như các định dạng giao dịch mới hơn, và cũng như là những dạng script mới.

Với nền tảng của đề tài này, em mong trong tương lai sẽ có nhóm tiếp tục phát triển đề tài lên thành một đề tài lớn hơn. Đồng thời, em cũng sẽ tiếp tục nghiên cứu về tiền mã hóa và tiếp tục phát triển ứng dụng để đạt được mục đích đặt ra của đề tài này, đồng thời cố gắng mang lại cho người dùng một trải nghiệm tốt nhất có thể.

Tài liệu tham khảo

- [1] J. Davis, "The Crypto-Currency: Bitcoin and its mysterious inventor," The New Yorker, [Online]. Available: <https://www.newyorker.com/magazine/2011/10/10/the-crypto-currency>.
- [2] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," [Online]. Available: <https://bitcoin.org/bitcoin.pdf>.
- [3] "Bitcoin core," [Online]. Available: <https://bitcoincore.org/>.
- [4] R. Skudnov, "Bitcoin Clients," Turku University of Applied Sciences, [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/47166/Skudnov_Rostislav.pdf?sequence=1.
- [5] "Bitcoin Core version 0.9.0 released," Bitcoin, [Online]. Available: <https://bitcoin.org/en/release/v0.9.0>.
- [6] "Libbitcoin: A C++ Bitcoin toolkit library for asynchronous apps," [Online]. Available: <https://libbitcoin.info/>.
- [7] "Armory Secure Wallet: The Only Open-Source Wallet With Cold Storage And Multi-Signature Support," [Online]. Available: <https://www.bitcoinarmory.com/>.
- [8] "Running A Full Node: Support the Bitcoin network by running your own full node," [Online]. Available: <https://bitcoin.org/en/full-node>.
- [9] "Unix build notes," Bitcoin, [Online]. Available: <https://github.com/bitcoin/bitcoin/blob/master/doc/build-unix.md>.
- [10] "Blockchain Size," Blockchain.com, [Online]. Available: <https://www.blockchain.com/charts/blocks-size>.
- [11] "libbitcoin-system," [Online]. Available: <https://github.com/libbitcoin/libbitcoin-system/blob/master/README.md>.
- [12] A. M. Antonopoulos, Mastering Bitcoin: Programming the Open Blockchain, O'Reilly Media, 2017.

- [13] "The History of Cryptography," [Online]. Available:
<https://cs.stanford.edu/people/eroberts/courses/soco/projects/public-key-cryptography/history.html>.
- [14] D. R. L. Brown, "Standards for Efficient Cryptography 2. SEC2: Recommended Elliptic Curve Domain Parameters," [Online]. Available:
<https://www.secg.org/sec2-v2.pdf>.
- [15] "BIP-0032: Hierarchical Deterministic Wallets," [Online]. Available:
https://en.bitcoin.it/wiki/BIP_0032.
- [16] "BIP-0039: Mnemonic code for generating deterministic keys," [Online]. Available: https://en.bitcoin.it/wiki/BIP_0039.
- [17] "BIP-0044: Multi-Account Hierarchy for Deterministic Wallets," [Online]. Available: https://en.bitcoin.it/wiki/BIP_0044.
- [18] "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation," [Online]. Available: <https://www.itu.int/itu-t/recommendations/rec.aspx?rec=X.680>.
- [19] "Transactions," [Online]. Available:
<https://developer.bitcoin.org/devguide/transactions.html>.
- [20] "Opt-in RBF FAQ," [Online]. Available:
https://bitcoincore.org/en/faq/optin_rbf/.
- [21] "Exodus + Trezor Hardware Wallet Experience: Advance security made easy," [Online]. Available: <https://www.exodus.com/trezor-wallet/>.