

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**NGÔ TRỌNG HIẾU
ĐINH TRUNG HẬU**

KHÓA LUẬN TỐT NGHIỆP

**ỨNG DỤNG NHẮN TIN NHÓM AN TOÀN
DỰA TRÊN GIAO THỨC MLS**

**SECURE GROUP MESSAGING APPLICATION
BASED ON MLS PROTOCOL**

KỸ SƯ NGÀNH AN TOÀN THÔNG TIN

TP. HỒ CHÍ MINH, 2019

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

**NGÔ TRỌNG HIẾU - 16520395
ĐINH TRUNG HẬU - 16520354**

KHÓA LUẬN TỐT NGHIỆP

**ỨNG DỤNG NHẮN TIN NHÓM AN TOÀN
DỰA TRÊN GIAO THỨC MLS**

**SECURE GROUP MESSAGING APPLICATION
BASED ON MLS PROTOCOL**

KỸ SƯ NGÀNH AN TOÀN THÔNG TIN

**GIẢNG VIÊN HƯỚNG DẪN
TS. PHẠM VĂN HẬU**

TP. HỒ CHÍ MINH, 2019

DANH SÁCH HỘI ĐỒNG BẢO VỆ KHÓA LUẬN

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo quyết định số .../QĐ-DHCNTT ...QĐ-DHCNTT ngày/.. của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

1. — Chủ tịch.
2. — Thư ký.
3. — Ủy viên.

LỜI CẢM ƠN

Nhóm thực hiện khóa luận chân thành cảm ơn thầy TS. Phạm Văn Hậu, cùng với ThS. Phan Thế Duy đã theo sát quá trình thực hiện đề tài, đóng góp những ý kiến thiết thực và hữu ích để nhóm có thể hoàn thành đề tài khóa luận một cách hoàn chỉnh nhất.

Nhóm xin gửi lời cảm ơn đến gia đình và bạn bè đã động viên, khuyến khích nhóm hoàn thành khoá luận.

Nhóm cũng xin cảm ơn đến quý thầy cô khoa Mạng máy tính và truyền thông, trường Đại học Công nghệ Thông tin - ĐHQG TP.HCM đã giúp đỡ và hỗ trợ nhóm.

Xin chân thành cảm ơn!

TP.Hồ Chí Minh, ngày tháng năm 2020
Nhóm tác giả

Mục lục

Danh sách hình vẽ	2
Danh sách bảng biểu	3
Danh sách thuật ngữ	4
Danh sách từ viết tắt	5
TÓM TẮT KHÓA LUẬN	6
1 MỞ ĐẦU	7
1.1 Vân đề đặt ra	7
1.2 Tính khoa học và tính mới của đề tài	11
1.2.1 Tính khoa học	11
1.2.2 Tính mới	11
1.3 Mục tiêu	11
1.4 Đối tượng nghiên cứu	11
1.5 Phạm vi nghiên cứu	12
2 TỔNG QUAN	13
2.1 Giới thiệu	13
2.2 Tổng quan Secure Two-party Messaging	14
2.3 From Two Parties to a Group	15
2.4 Kiến thức nền tảng	16
2.4.1 Giao thức Messaging Layer Security (MLS)	16
2.5 Ratchet Tree	20
2.5.1 Thuật ngữ tính toán trong cây	20
2.5.2 Ratchet Tree Nodes	22
2.6 Basic background on TreeKEM	22

2.6.1	Key Update	23
2.6.2	Init and Application Secrets	23
2.6.3	Adding and Removing Users	23
3	PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG	24
3.1	Giới thiệu hệ thống	24
3.2	Quy trình gây quỹ cộng đồng	24
3.2.1	Các đối tượng trong quy trình	24
3.2.2	Sơ đồ quy trình gây quỹ	25
3.3	Kiến trúc hệ thống	26
3.4	Các chức năng	27
3.4.1	Tổng quan các chức năng	27
3.4.2	Chức năng nộp tiền và rút tiền	27
3.4.2.1	Mục tiêu	27
3.4.2.2	Cách thức hoạt động	30
3.4.3	Chức năng quản lý định danh	32
3.4.3.1	Mục tiêu	32
3.4.3.2	Cách hoạt động	32
3.4.4	Tạo lập và lưu trữ chiến dịch gây quỹ	34
3.4.4.1	Mục tiêu	34
3.4.4.2	Cách thức hoạt động	35
3.4.5	Chức năng giải ngân	37
3.4.5.1	Mục tiêu	37
3.4.5.2	Cách thức hoạt động	37
3.4.6	Hoàn tiền chiến dịch gây quỹ	38
3.4.6.1	Mục tiêu	38
3.4.6.2	Cách hoạt động	39
3.5	Tổ chức dữ liệu	40
3.5.1	Dữ liệu phi tập trung	40
3.5.2	Dữ liệu tập trung	40
4	HÌNH THỰC VÀ ĐÁNH GIÁ HỆ THỐNG	43
4.1	Hiện thực	43
4.1.1	Môi trường hiện thực	43
4.1.2	Các công nghệ được sử dụng	43
4.1.2.1	Công nghệ ReactJS/NodeJS	44
4.1.2.2	Material UI framework	45

4.1.2.3	Cơ sở dữ liệu Redis	46
4.1.2.4	IPFS	47
4.1.2.5	Bộ công cụ Truffle framework	47
4.1.2.6	Thư viện web3js	48
4.1.2.7	Ví Metamask	48
4.1.3	Các bước hiện thực	48
4.1.3.1	Cấu hình thông số các service	50
4.1.3.2	Tạo các service	51
4.1.3.3	Chạy các service	53
4.1.4	Kết quả hiện thực	53
4.2	Đánh giá hệ thống đã hiện thực	58
4.2.1	Kiểm tra các quy trình trong hệ thống	58
4.2.1.1	Mục tiêu	58
4.2.1.2	Phương pháp thực hiện	59
4.2.1.3	Kết quả thực hiện	61
4.2.2	Đo lường tốc độ thực hiện giao dịch	62
4.2.2.1	Môi trường thực hiện đánh giá	62
4.2.2.2	Phương pháp thực hiện đánh giá	63
4.2.2.3	Kết quả đánh giá	64
4.2.3	Chi phí thực hiện các giao dịch trong hệ thống	64
4.2.3.1	Môi trường thực hiện đánh giá	64
4.2.3.2	Phương pháp thực hiện đánh giá	65
4.2.3.3	Kết quả đo lường chi phí các giao dịch trong hệ thống	65
4.2.4	Phân tích bảo mật của hợp đồng thông minh trong hệ thống	68
4.2.4.1	Các lỗ hổng phổ biến trong hợp đồng thông minh trên Ethereum	68
4.2.4.2	Các công cụ phân tích bảo mật smart contract	71
4.2.4.3	Kết quả phân tích	72
4.2.5	Đánh giá tốc độ tải trang của giao diện người dùng	72
4.2.5.1	Môi trường thực hiện	72
4.2.5.2	Kết quả đánh giá	73
5	KẾT LUẬN	76
5.1	Kết quả đạt được	76
5.2	Ưu điểm và khuyết điểm của hệ thống	76
5.2.1	Ưu điểm	76
5.2.2	Khuyết điểm	77

5.3	Khó khăn	77
5.4	Hướng phát triển	78
TÀI LIỆU THAM KHẢO		79
A	Mã hợp đồng thông minh - Wallet	79
B	Mã hợp đồng thông minh - Campaigns	82
C	Mã hợp đồng thông minh - Identity	90
D	Mã hợp đồng thông minh - Disbursement	95

Danh sách hình vẽ

1.1	Mô hình mã hóa end to end encryption	8
1.2	Secure Real-time Transport Protocol (SRTP)	9
1.3	Z RTP (gồm Z và Real-time transport Protocol)	9
1.4	Mô hình kiến trúc cơ bản của giao thức MLS	10
2.1	Mã khóa riêng tư tin nhắn trong ứng dụng Whatsapp	14
2.2	Quy trình thêm Client vào nhóm	18
2.3	Quy trình cập nhật key của nhóm	19
2.4	Quy trình xóa thành viên nhóm	19
2.5	Direct path của C là (CD, ABCD, ABCDEFG).	21
2.6	Resolution của node 5 là danh sách [CD, C]	22
3.1	Sơ đồ quy trình gây quỹ cộng đồng	25
3.2	Sơ đồ kiến trúc hệ thống	26
3.3	Sơ đồ tổng quan các chức năng trong hệ thống	28
3.4	Sơ đồ phân rã các chức năng trong hệ thống	28
3.5	Sơ đồ tổng quan các đối tượng và luồng tương tác dữ liệu	29
3.6	Sơ đồ hoạt động chức năng nộp tiền	30
3.7	Sơ đồ hoạt động chức năng rút tiền	31
3.8	Sơ đồ cách thức lưu trữ hồ sơ định danh	33
3.9	Sơ đồ cách thức chia sẻ thông tin định danh	34
3.10	Sơ đồ hoạt động tiến trình tạo lập chiến dịch	36
3.11	Kiến trúc hợp đồng thông minh	41
3.12	Thiết kế cơ sở dữ liệu tập trung	42
4.1	Sơ đồ hiện thực hệ thống	44
4.2	Biểu đồ thể hiện thái độ người dùng khi không được sử dụng thư viện Material	46
4.3	Màn hình giao diện trang nộp tiền và rút tiền	53
4.4	Màn hình giao diện trang thêm nhân viên xác minh	54

4.5	Màn hình giao diện trang đăng ký định danh	54
4.6	Màn hình hộp thoại xác nhận kí transaction trên Metamask	55
4.7	Ảnh chụp chi tiết transaction trên etherscan	55
4.8	Giao diện trang hiển thị danh sách hồ sơ định danh	56
4.9	Màn hình hiển thị thông tin định danh người dùng	56
4.10	Giao diện trang đăng ký chiến dịch gây quỹ	57
4.11	Giao diện người dùng khi xem thông tin chiến dịch đang chờ xét duyệt	58
4.12	Giao diện của trang danh sách các chiến dịch	58
4.13	Giao diện của trang chi tiết thông tin chiến dịch	59
4.14	Kết quả kiểm thử kịch bản 1	62
4.15	Kết quả kiểm thử kịch bản 2	62
4.16	Biểu đồ thể hiện tốc độ thực hiện giao dịch giữa các hàm	65
4.17	Ảnh chụp màn hình kết quả đo lường chi phí giao dịch	67
4.18	Kết quả phân tích hợp đồng thông minh với Remix IDE	73
4.19	Kết quả phân tích hợp đồng thông minh với Slither	74
4.20	Kết quả phân tích hợp đồng thông minh với Smartcheck	75

Danh sách bảng biểu

4.1	Các hàm và tham số đầu vào được dùng để đo thời gian thực hiện giao dịch	63
4.2	Bảng kết quả đánh giá thời gian hiện thực một số hàm trong hệ thống.	64
4.3	Các hàm và dữ liệu đầu vào được dùng để đo lường chi phí giao dịch	66
4.4	Kết quả đo lường chi phí giao dịch	67
4.5	Kết quả đo lường chi phí triển khai các hợp đồng	68
4.6	Bảng tổng hợp các lỗ hổng trong hợp đồng thông minh trên Ethereum	68
4.7	Bảng kết quả đo tốc độ truy xuất front-end của hệ thống	73

Danh sách thuật ngữ

Thuật ngữ	Điễn giải
address	địa chỉ, địa chỉ người dùng trong mạng blockchain. 32
back-end	lập trình phía máy chủ, xử lý luồng thông tin giao diện người dùng. 44
blockchain	chuỗi khôi. 6, 24, 27, 32, 33, 35, 39, 47, 62, 76
cryptocurrency	đồng tiền mã hóa. 25, 26
ether	đồng tiền mã hóa trong ethereum. 30, 32, 69
front-end	hệ thống các giao diện người dùng, tương tác trực tiếp với người dùng. 43, 44, 49
service	dịch vụ. 49, 50
smart contract	hợp đồng thông minh. 44
transaction	giao dịch. 27, 30, 31, 36, 54, 62, 63, 65
Trusted Computing Base	Cơ sở tính toán đáng tin cậy. 13

Danh sách từ viết tắt

Từ viết tắt	Từ đầy đủ
ABI	Application Binary Interface. 48, 49
IPFS	InterPlanetary File System. 33, 44, 47
P2P	Peer-to-Peer. 47, 48
RPC	Remote Procedure Call. 48

TÓM TẮT KHÓA LUẬN

Kết quả đạt được bao gồm:

- Đề xuất mô hình ứng dụng gây quỹ cộng đồng từ thiện dựa trên công nghệ blockchain cải thiện các vấn đề hiện tại của ứng dụng gây quỹ từ thiện.
- Hiện thực mô hình đã đề xuất với các chức năng cơ bản của ứng dụng gây quỹ từ thiện cộng đồng như: tạo chiến dịch, đóng góp vào chiến dịch, giải ngân. Ngoài ra hệ thống còn hiện thực các chức năng nổi bật mà các hệ thống hiện tại chưa hoàn thiện như: lưu trữ và quản lý thông tin định danh, hoàn tiền tự động khi chiến dịch không đạt được mục tiêu, giải ngân và bỏ phiếu giải ngân nhiều giai đoạn.

Cấu trúc của bài báo cáo này như sau:

- **Chương 1: Mở đầu** – nêu ra vấn đề cần giải quyết, tính khoa học và tính mới của đề tài. Đề ra mục tiêu cũng như đối tượng và phạm vi nghiên cứu đề tài.
- **Chương 2: Tổng quan** – giới thiệu tổng quan về công nghệ, các nghiên cứu liên quan cùng kiến thức nền tảng về công nghệ.
- **Chương 3: Phân tích và thiết kế hệ thống** – phần này tập trung trình bày thiết kế mô hình hệ thống và các chức năng trong hệ thống.
- **Chương 4: Hiện thực và đánh giá hệ thống** – .
- **Chương 5: Kết luận** – trình bày kết quả đạt được, ưu điểm, nhược điểm của hệ thống. Từ đó đề ra hướng phát triển của đề tài.

Chương 1

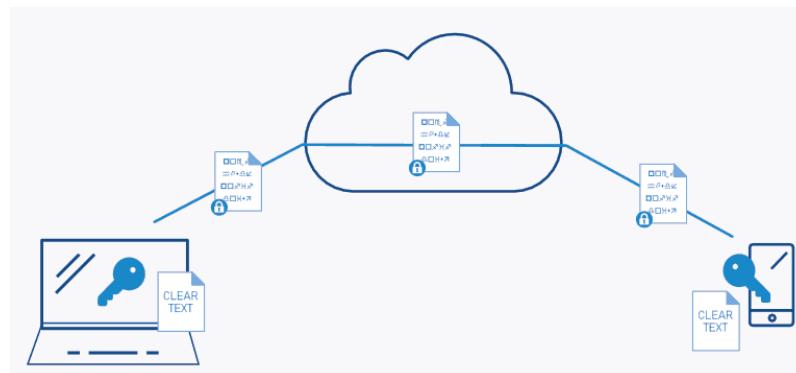
MỞ ĐẦU

1.1 Vấn đề đặt ra

Trong xã hội công nghệ thông tin đóng vai trò quan trọng hiện nay, tất cả những thông tin cá nhân của chúng ta đều có khả năng bị theo dõi do những hacker xâm nhập và đánh cắp. Đối với một người dùng bình thường, những thông tin đó đôi khi chỉ là những tin nhắn, dòng chat, tài liệu thông thường, nhưng ở mức độ cao hơn điều này gây ra hậu quả vô cùng nghiêm trọng đối với các công ty, tập đoàn do những thông tin mật nếu bị tiết lộ ra ngoài sẽ gây thiệt hại rất lớn. Do đó, nếu như những dữ liệu quan trọng của bạn được bảo mật và mã hóa, sẽ rất khó để hacker có thể theo dõi và đánh cắp được. Việc mã hóa dữ liệu, đơn giản là việc tăng thêm một lớp bảo mật cho dữ liệu bằng cách chuyển đổi dữ liệu sang một dạng khác thông qua một mã khóa với những quy tắc tùy biến. Vì vậy, kể cả khi dữ liệu của bạn có bị đánh cắp, việc giải mã dữ liệu cũng là rất khó khăn. Một ví dụ đơn giản cho việc mã hóa dữ liệu: Nếu như bạn chỉ đặt mật khẩu cho máy tính, laptop của bạn, hacker chỉ cần một vài thủ thuật để bẻ qua lớp mật khẩu là có thể truy cập được dữ liệu, hoặc đơn giản chỉ là cắm thiết bị lưu trữ sang một hệ thống khác, tuy nhiên nếu như dữ liệu được mã hóa, kể cả khi có được dữ liệu rồi cũng rất khó để giải mã được như ban đầu nếu không có mã khóa.

Ngày nay với sự ra đời của thế giới kỹ thuật số, việc giao tiếp của con người đã trở thành một trong những khía cạnh thiết yếu nhất của cuộc sống, dễ dàng và nhanh chóng hơn. Ta có thể dễ dàng thực hiện giao tiếp thời gian thực với bạn bè và gia đình ở bất kỳ nơi nào trên thế giới thông qua các ứng dụng trò chuyện, email và các công cụ dựa trên web phổ biến khác. Ngày càng có nhiều người sử dụng các ứng dụng điện thoại thông minh để trò chuyện, và theo đó, nguy cơ nội dung tin nhắn bị tin tặc nghe lén càng tăng cao. Nếu chỉ sử dụng mã hóa khi truyền tải không thì sẽ rất nguy hiểm khi mà nội dung trò truyền có thể rò rỉ nếu server bị hacker xâm nhập do dữ liệu chỉ mã hóa trên đường truyền chứ không mã hóa khi lưu vào cơ

sở dữ liệu. Do đó, Facebook, Whatsapp, Viber hay các nền tảng nhắn tin khác đã và đang áp dụng phương thức mã hóa End-to-end Encryption (E2EE) trong ứng dụng của họ. E2EE là phương thức mã hóa mà chỉ duy nhất những người giao tiếp với nhau có thể hiểu được thông điệp được mã hóa. Điều này đồng nghĩa với việc kể cả chủ sở hữu của kênh truyền tải dữ liệu, những nhà cung cấp dịch vụ Internet hay hacker cũng khó có thể biết được những thông tin người giao tiếp đang truyền tải. Phương thức mã hóa này sử dụng mã khóa (key) thuộc về những người đang trực tiếp liên lạc và truyền tải thông tin, dữ liệu, và nếu không có mã khóa này, không một bên thứ ba nào có thể giải mã dữ liệu.

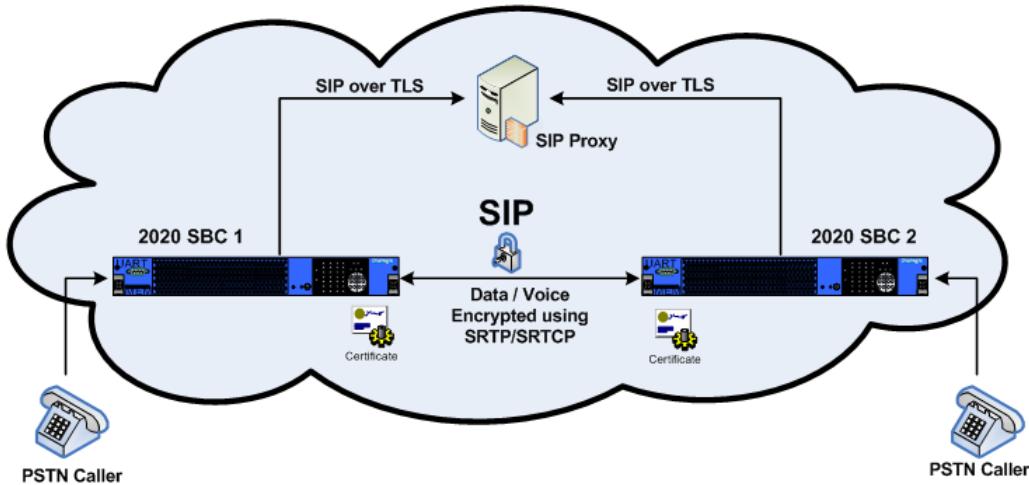


Hình 1.1: Mô hình mã hóa end to end encryption

Khi thế giới chuyển sang mã hóa đầu cuối cho các nền tảng nhắn tin cá nhân, các doanh nghiệp cũng đang tích hợp chúng mức độ bảo mật trong các ứng dụng nhắn tin của công ty. Giao thức Signal [1], được triển khai cho cơ sở một tỷ người dùng của WhatsApp vào năm 2016, là giao thức mã hóa đầu cuối đầu tiên được triển khai trên toàn cầu. Đây là một giao thức mật mã không liên kết, có thể được sử dụng để cung cấp mã hóa đầu cuối cho các cuộc gọi thoại, cuộc gọi video và các cuộc hội thoại nhắn tin tức thời. Signal với tiêu chí hoạt động là đảm bảo tính riêng tư tối đa cho người dùng, dịch vụ này không lưu trữ bất kỳ dữ liệu gì của khách hàng, kể cả dữ liệu đã mã hóa. Ngoài ra còn có các giao thức như Secure Real-time Transport Protocol (SRTP) [2], Elliptic-curve Diffie–Hellman [3], ZRTP (gồm Z và Real-time transport Protocol)... cũng là các giao thức cung cấp mã hóa đầu cuối, thành lập kênh trò chuyện bí mật thông qua phương pháp trao đổi khóa Diffie–Hellman.

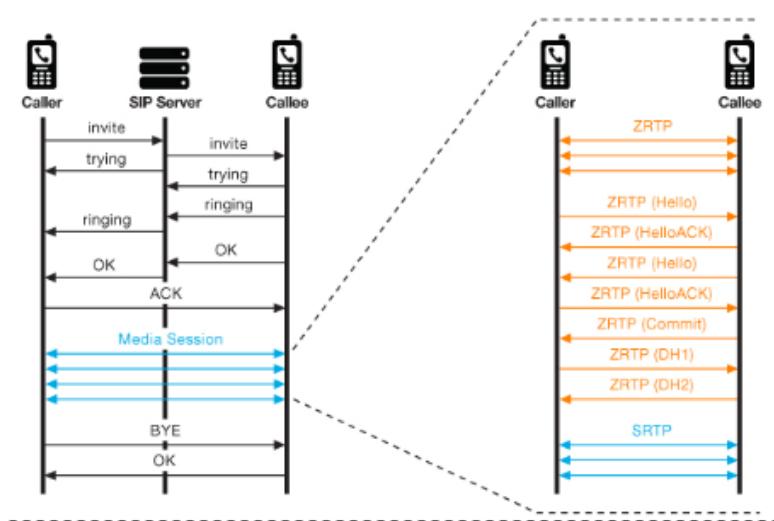
SRTP là cấu hình giao thức Real-time Transport Protocol, nhằm cung cấp mã hóa, xác thực và tính toàn vẹn của tin nhắn và bảo vệ dữ liệu RTP khỏi replay attack trong cả ứng dụng unicast và multicast. Nó được phát triển bởi một nhóm nhỏ các Internet Protocol và các chuyên gia mật mã từ Cisco và Ericsson.

ZRTP là một giao thức trao đổi khóa mật mã để đàm phán các khóa để mã hóa giữa hai điểm cuối trong cuộc gọi điện thoại thông qua Voice over Internet Protocol (VoIP) dựa trên giao



Hình 1.2: Secure Real-time Transport Protocol (SRTP)

thức Real-time Transport Protocol. Nó sử dụng trao đổi khóa của Diffie-Hellman và giao thức SRTP để mã hóa.

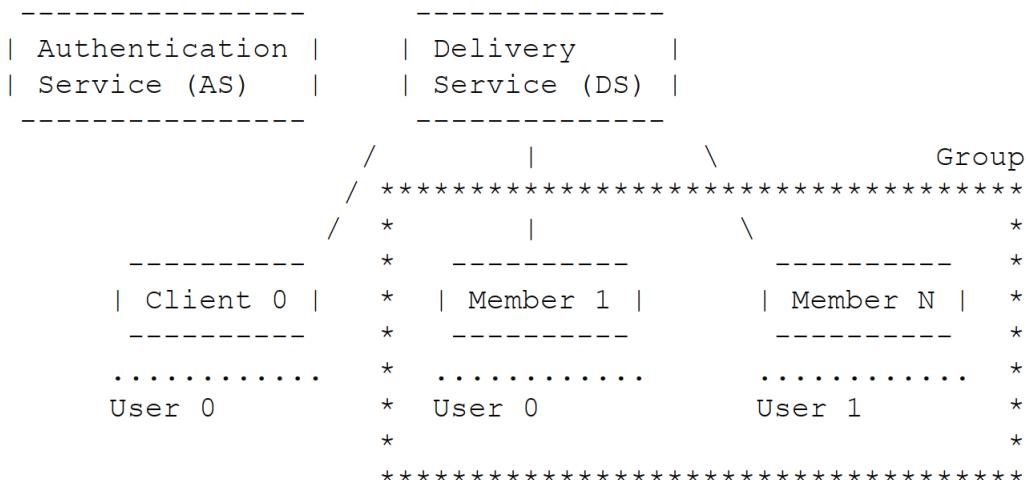


Hình 1.3: ZRTP (gồm Z và Real-time transport Protocol)

Các giao thức trên đều có ưu và nhược điểm riêng, tuy nhiên, chúng đều có chung một nhược điểm là đều cung cấp một giao thức thỏa thuận khóa bí mật để mã hóa thông tin chỉ giữa hai điểm đầu cuối. Tức là mã hóa đầu cuối hiện tại chỉ hỗ trợ trao đổi thông tin giữa hai người duy nhất và không hỗ trợ khi có thêm người thứ ba. Điều này là đúng với các ứng dụng phổ biến hiện nay như Viber, Telegram, hay thậm chí là Facebook Messenger. Các ứng dụng nói trên đều hỗ trợ trò chuyện riêng tư thông qua mã hóa đầu cuối, nhưng chỉ hỗ trợ giữa hai thành viên với nhau. Wire [4], một ứng dụng hiếm hoi ít người biết đến, là ứng dụng hiện tại có hỗ trợ mã hóa đầu cuối cho việc trao đổi thông tin nhóm. Tuy nhiên, thực tế cho thấy cốt

lỗi của việc mã hóa đầu cuối trong nhóm mà Wire sử dụng vẫn chỉ là mã hóa giữa hai người với nhau [5].

Để giải quyết khó khăn trên, Lực lượng Chuyên trách về Kỹ thuật Liên mạng (IETF) đã và đang xây dựng lên giao thức Messaging Layer Security (MLS) [6] – một chuẩn giao thức chung hiện đại, an toàn bảo mật cho việc nhắn tin nhóm. MLS là một lớp bảo mật để mã hóa tin nhắn trong nhóm có số lượng thành viên lớn. Mục đích MLS hướng tới xây dựng một chuẩn giao thức chung, có thể ứng dụng đa nền tảng, hỗ trợ mã hóa tin nhắn trong nhóm chứa đến 50000 thành viên và đảm bảo an ninh bảo mật thông tin. Đáp ứng đầy đủ yêu cầu bảo mật như: Forward Secrecy (FS) – Trong mật mã còn được gọi là Perfect Forward Secrecy (FFS), là một tính năng của giao thức thỏa thuận khóa cụ thể mang lại sự đảm bảo rằng các khóa phiên sẽ không bị xâm phạm ngay cả khi khóa riêng của máy chủ bị xâm phạm. Post-compromise security (PCS) – Kể từ khi khóa riêng của máy chủ bị xâm phạm vẫn đảm bảo rằng các khóa phiên của người dùng trong tương lai sẽ không bị xâm phạm.



Hình 1.4: Mô hình kiến trúc cơ bản của giao thức MLS

Xuất phát từ hạn chế trên và từ lợi ích có được khi sử dụng giao thức MLS, đề tài này hướng tới việc tạo nên một kênh giao tiếp bí mật, truyền tải thông tin nhạy cảm một cách riêng tư và chỉ có người liên quan mới có thể đọc được. Kết quả của nghiên cứu này sẽ nâng cao tính an toàn, bảo mật cho dữ liệu, tránh những trường hợp thất thoát thông tin dữ liệu quan trọng khi giao tiếp thông qua internet.

1.2 Tính khoa học và tính mới của đề tài

1.2.1 Tính khoa học

Nghiên cứu này sẽ tập trung nghiên cứu vấn đề mã hóa đầu cuối trong khi nhắn tin nhóm dựa trên MLS protocol. Nghiên cứu chi tiết cách hoạt động và triển khai giao thức MLS. Nghiên cứu cơ chế hoạt động của các thuật toán mã hóa đầu cuối, kết hợp các thuật toán vào xây dựng một ứng dụng nhắn tin nhóm đáp ứng yêu cầu số lượng thành viên lớn và đạt hiệu suất cao.

Đề tài này hướng đến việc xây dựng ứng dụng nhắn tin nhóm an toàn, riêng tư, bí mật. Ứng dụng này có các chức năng cơ bản cho việc nhắn tin bí mật giữa nhiều thành viên trong nhóm với nhau, đảm bảo tính riêng tư, bí mật thông qua áp dụng mã hóa đầu cuối, đáp ứng nhu cầu trao đổi thông tin nhạy cảm cho công ty, doanh nghiệp có số lượng nhân viên lớn. Nếu nghiên cứu thành công, thì đây sẽ là ứng dụng đầu tiên áp dụng mã hóa đầu cuối vào việc trao đổi thông tin giữa nhiều cá nhân, đáp ứng nhu cầu bảo đảm bí mật thông tin trao đổi khỏi bị theo dõi hay bị đánh cắp cho các công ty, doanh nghiệp.

1.2.2 Tính mới

Trên thế giới, số công trình nghiên cứu ứng dụng mã hóa đầu cuối vào việc trao đổi thông tin trong nhóm ở thực tiễn hầu như không có hoặc đang nghiên cứu nhưng không công khai. Từ đó, vấn đề nghiên cứu này sẽ đáp ứng nhu cầu nhắn tin nhóm cho các doanh nghiệp có yêu cầu cao trong việc bảo vệ sự riêng tư, bí mật, an toàn.

1.3 Mục tiêu

Xây dựng ứng dụng nhắn tin nhóm cơ bản dựa trên MLS protocol có các thuộc tính cơ bản như trên. Ứng dụng có các chức năng chính:

- Tạo nhóm.
- Xóa nhóm.
- Thêm người dùng vào nhóm.
- Xóa người dùng khỏi nhóm.

1.4 Đối tượng nghiên cứu

- Giao thức MLS.
- Các thuật toán liên quan đến mã hóa đầu cuối như SRTP, ZRTP, ...

1.5 Phạm vi nghiên cứu

Ứng dụng có các chức năng cơ bản cho việc trao đổi thông tin bằng văn bản.

Chương 2

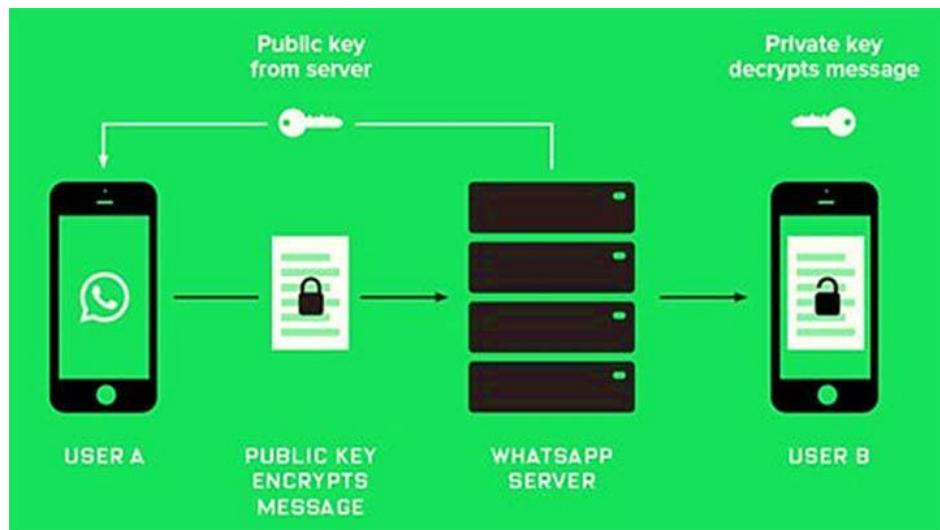
TỔNG QUAN

2.1 Giới thiệu

Mã hóa đầu cuối (E2E) là một dạng mã hóa dữ liệu đầu cuối đối với những tin nhắn chat, thư email được gửi và trao đổi trên Internet. E2E đang trở thành tiêu chuẩn cho các ứng dụng nhắn tin. WhatsApp là một ví dụ nổi bật, với hàng tỷ người dùng. Tất cả những trao đổi trực tiếp trên ứng dụng đó đều được mã hóa bảo vệ như: “Tin nhắn, hình ảnh, cuộc gọi, video, âm thanh, và một vài tệp tin đính kèm khác”.

Mã hóa đầu cuối cải thiện sự riêng tư của người dùng bằng cách làm cho tin nhắn không thể đọc được cho bất kỳ ai ngoài người nhận dự định của họ. Đặc biệt, tin nhắn không thể đọc được từ máy chủ nhắn tin. Điều này làm giảm Trusted Computing Base và bảo vệ chống lại khả năng máy chủ có thể bị xâm phạm, ví dụ như nhân viên lừa đảo, tin tặc hoặc cơ quan giám sát. Các sự kiện trong thế giới thực cho thấy sự thỏa hiệp của máy chủ là mối đe dọa hợp pháp, bao gồm cả việc truy cập hàng loạt NSA vào các công ty Internet dữ liệu qua PRISM [7] và bằng chứng về việc nhân viên truy cập dữ liệu không phù hợp tại Facebook [8] và Snap [9].

Tiện ích mới cho người dùng các ứng dụng là có 2 giao diện mã khóa: mã khóa công khai và mã khóa riêng tư. Khóa công khai sẽ cho phép mọi người nhìn thấy thông tin dữ liệu trao đổi, và đăng tải lên đám mây Google. Ngược lại, khóa riêng tư thì chỉ cho phép người nhận và người gửi xem thông tin nội dung dữ liệu truyền tải, không cho bất kỳ bên thứ 3 nào tham gia quá trình trao đổi này. Mọi thông tin bảo mật đều được khóa riêng tư lưu trữ trên trình duyệt. Để nâng cấp thêm tính năng này, cho phép người dùng có thể điều chỉnh thiết lập lại thời gian chờ của tin nhắn, hạn chế lưu trữ các tin nhắn đến, tự động xóa các tin nhắn cũ, theo quy định số lượng tin nhắn hay thời gian tin nhắn ra khỏi bộ nhớ hệ thống. Cũng có thể lưu trữ một lượng lớn tin nhắn.



Hình 2.1: Mã khóa riêng tư tin nhắn trong ứng dụng Whatsapp

2.2 Tổng quan Secure Two-party Messaging

Trong mục này, chúng tôi mô tả ngắn gọn trường hợp đơn giản hơn về nhắn tin hai bên an toàn. Chúng tôi cũng nêu rõ những khó khăn khi chuyển sang nhắn tin nhóm an toàn. Chúng tôi sẽ giới thiệu các khái niệm mật mã làm nền cho các chương sau. Định nghĩa của chúng tôi theo Unger et al. [8, §V].

Mục tiêu cơ bản của nhắn tin bảo mật hai bên là cho phép hai người dùng có một cuộc trò chuyện an toàn. Bảo mật trong bối cảnh này có nhiều phần. Bảo mật (Confidentiality) là thuộc tính mà chỉ có hai người tham gia có thể biết được tin nhắn plaintext. Tính toàn vẹn (Integrity) là thuộc tính mà chỉ hai người tham gia mới có thể gửi tin nhắn, tức là, những người tham gia sẽ từ chối các tin nhắn đã được gửi hoặc giả mạo bởi người ngoài. Xác thực (Authentication) là thuộc tính mà hai người tham gia đồng ý về các danh tính của nhau, thường được biểu thị bằng các khóa công khai.

Các thuộc tính này thường được đo dựa trên hacker, có thể đọc và lưu trữ tất cả các liên lạc giữa các bên, gửi tin nhắn giả mạo và bắt đầu các cuộc hội thoại an toàn song song, tuân theo giới hạn thực tế về sức mạnh tính toán của nó.

Mã hóa bất đối xứng, như trong Pretty Good Privacy (PGP), đạt được các thuộc tính trên chống lại một hacker. Tuy nhiên, nó không đạt được các đặc tính bảo mật nâng cao về bảo mật bí mật và bảo mật sau thỏa hiệp, được đo lường trước một kẻ thù cũng có thể thỏa hiệp với người dùng, học tất cả các khóa bí mật của họ tại một thời điểm nhất định.

Bảo mật chuyển tiếp (Forward secrecy) là thuộc tính mà một hacker thỏa hiệp một số hoặc tất cả người dùng sẽ không thể giải mã tin nhắn từ trước khi thỏa hiệp. Có thể đạt được bí mật

về mức độ chi tiết của toàn bộ các cuộc hội thoại cùng với các thuộc tính trên bằng cách sử dụng trao đổi Diffie-Hellman đã được xác thực để thiết lập khóa chia sẻ tồn tại ngắn và mới (không được sử dụng trước đó) cho mỗi cuộc hội thoại, sau đó mã hóa tin nhắn theo khóa sử dụng sơ đồ mã hóa xác thực với dữ liệu liên kết (AEAD) [9], như trong TLS-DHE [10]. Bảo mật chuyển tiếp ở mức độ chi tiết của các thông điệp riêng lẻ có thể đạt được bằng cách sử dụng thêm tính năng ghép mã khóa xác định (deterministic key ratcheting), được giới thiệu bởi SCIMP [11]. Theo cách tiếp cận này, người dùng mã hóa mọi tin nhắn bằng một khóa duy nhất, xuất phát từ một số khóa được chia sẻ bằng cách sử dụng chuỗi ứng dụng hàm băm và xóa khóa ngay sau khi sử dụng.

Bảo mật sau thỏa hiệp (Post-compromise security), còn được gọi là bảo mật ngược hoặc tự phục hồi, là thuộc tính mà hacker tạm thời thỏa hiệp một số hoặc tất cả người dùng không thể giải mã tin nhắn vô thời hạn sau khi thỏa hiệp được phục hồi. Điều này hữu ích trong trường hợp đối thủ tạm thời truy cập thiết bị và tạo một bản sao trạng thái của thiết bị, ví dụ như thông qua phần mềm độc hại hoặc kiểm tra vật lý, nhưng sau đó mất quyền truy cập vào thiết bị. Bảo mật sau thỏa hiệp có thể đạt được bằng cách liên tục làm mới khóa chia sẻ tồn tại ngắn bằng cách sử dụng các trao đổi Diffie-Hellman mới, một kỹ thuật được giới thiệu bởi OTR [12].

Tất cả các thuộc tính trên đều đạt được bằng giao thức two-party Signal [13, 14], được sử dụng cho các cuộc hội thoại hai bên trong Signal và WhatsApp. Giao thức Signal không đồng bộ ở chỗ người dùng có thể gửi tin nhắn khi người dùng khác ngoại tuyến, sử dụng máy chủ lưu trữ và chuyển tiếp để đệm tin nhắn. Điều này đúng ngay cả với tin nhắn đầu tiên trong một cuộc trò chuyện: bằng cách tải xuống một người dùng khác làm con mồi từ một máy chủ không tin cậy, người dùng có thể thiết lập một khóa chia sẻ tồn tại ngắn xác thực mà không cần giao tiếp. Ở đây một con mồi (prekeys) là thông điệp đầu tiên của một cuộc trao đổi Diffie-Hellman đã được chứng thực; mỗi người dùng tải lên trước một số con mồi cho một máy chủ không tin cậy, để những người dùng khác sau đó có thể sử dụng chúng để hoàn thành trao đổi Diffie-Hellman và bắt đầu một cuộc trò chuyện ngay lập tức. Giao thức Signal cũng đạt được thuộc tính từ chối, điều này nói rõ rằng một người dùng không thể chứng minh bằng mật mã rằng người dùng kia đã gửi bất kỳ tin nhắn cụ thể nào.

2.3 From Two Parties to a Group

Chuyển từ cài đặt hai bên sang các nhóm kích thước tùy ý đưa ra những thách thức mới. Một cách tiếp cận đơn giản để đạt được các thuộc tính bảo mật trên là cho người dùng gửi tin nhắn nhóm qua các kênh giao thức Signal riêng biệt cho từng người dùng, như trong các cuộc trò chuyện nhóm Signal [15]; tuy nhiên, điều này không hiệu quả, vì người gửi phải mã hóa và gửi từng tin nhắn một lần cho mỗi thành viên nhóm. Đạt được các thuộc tính như bảo mật

sau thỏa hiệp trong khi vẫn cho phép truyền phát thông điệp là một thách thức không cần thiết.

Ngoài ra, việc cài đặt nhóm sẽ thêm các thuộc tính bảo mật mong muốn mới. Xác thực tác giả là thuộc tính mà người dùng có thể xác minh ai đã gửi một tin nhắn cụ thể. Tính toàn vẹn và xác thực ngụ ý rằng người dùng luôn biết rằng một số thành viên nhóm dự định đã gửi tin nhắn, nhưng không giống như trong trường hợp hai bên, họ không biết đó là thành viên nhóm nào, ngoại trừ đó là ai đó ngoài chính họ. Tính nhất quán là thuộc tính mà tất cả người dùng đồng ý về những gì đang xảy ra trong nhóm. Điều này bao gồm tính nhất quán của người tham gia, trong đó tất cả người dùng đồng ý về nhóm thành viên nhóm và tính nhất quán của bảng điểm, trong đó tất cả người dùng đồng ý về nhóm thư được gửi cho đến nay và thứ tự của họ. Lưu ý rằng các hacker cho các thuộc tính này có thể bao gồm các thành viên nhóm độc hại, những người có thể cố gắng giả mạo các tác giả tin nhắn hoặc phá vỡ tính nhất quán của người dùng khác, ngoài các hacker, có thể bao gồm cả máy chủ.

Trật tự tin nhắn cũng khó khăn hơn trong cài đặt nhóm. Thông thường, các ứng dụng nhắn tin hiển thị tin nhắn theo thứ tự tuyến tính. Tuy nhiên, khi có nhiều hơn hai người dùng, thật khó để thực thi một trật tự tuyến tính nhất quán mà không có máy chủ trung tâm và không phải tất cả các ứng dụng đều yêu cầu một thứ tự tuyến tính. Vì vậy, chúng tôi cũng xem xét hai thứ tự yêu hơn. Thứ tự trước sau là thứ tự từng phần trong đó m đứng trước m' nếu tác giả của m nhận và xử lý m trước khi gửi m'. Điều này bao gồm trường hợp m và m' có cùng tác giả và m được gửi trước m'. Thứ tự trước sau chính xác là những gì CRDT yêu cầu: để thiết lập trạng thái chia sẻ nhất quán, tất cả người dùng phải nhận tin nhắn theo thứ tự trước sau nhất quán, nhưng họ không cần nhận tin nhắn theo cùng một thứ tự tuyến tính [5]. Chúng tôi cũng định nghĩa thứ tự cho mỗi người gửi là thứ tự từng phần trong đó m đứng trước m' nếu m và m' có cùng tác giả và tác giả đã gửi m trước m'.

2.4 Kiến thức nền tảng

2.4.1 Giao thức Messaging Layer Security (MLS)

Mục tiêu của giao thức này là cho phép một nhóm người tham gia trao đổi tin nhắn bí mật và xác thực. Bằng cách lấy ra một chuỗi các bí mật và Key chỉ được biết bởi các thành viên nhóm. Điều này nên được giữ bí để chống lại một kẻ thù mạng và nên giữ bí mật cả về phía trước và sau thỏa hiệp đối với sự thỏa hiệp của người tham gia.

Chúng tôi mô tả thông tin được lưu trữ bởi mỗi client là `_state_`, bao gồm cả dữ liệu công khai và riêng tư. Trạng thái ban đầu được thiết lập bởi người tạo nhóm, đó là một nhóm chỉ chứa chính họ. Sau đó, người tạo sẽ gửi các yêu cầu `_Add_` cho mỗi client trong nhóm thành viên ban đầu, theo sau là thông báo `_Commit_` kết hợp tất cả `_Adds_` vào trạng thái nhóm.

Cuối cùng, người tạo nhóm tạo ra một thông báo _Welcome_ tương ứng với _Commit_ và gửi trực tiếp thông báo này đến tất cả các thành viên mới, những người có thể sử dụng thông tin chứa trong đó để thiết lập trạng thái nhóm của riêng họ và lấy bí mật chung. Thành viên trao đổi thông điệp _Commit_ bảo mật sau thỏa hiệp, để thêm thành viên mới và xóa thành viên hiện có. Các thông điệp này tạo ra các bí mật được chia sẻ mới có mối liên hệ nhân quả với người tiền nhiệm của chúng, tạo thành một Biểu đồ chu kỳ có hướng (Directed Acyclic Graph - DAG) hợp lý.

Các thuật toán giao thức chúng tôi chỉ định ở đây. Mỗi thuật toán chỉ định cả (i) cách một client thực hiện thao tác và (ii) cách các client khác cập nhật trạng thái của họ dựa trên nó:

- Thêm một thành viên, được khởi xướng bởi một thành viên hiện tại;
- Cập nhật bí mật của một thành viên;
- Xóa một thành viên.

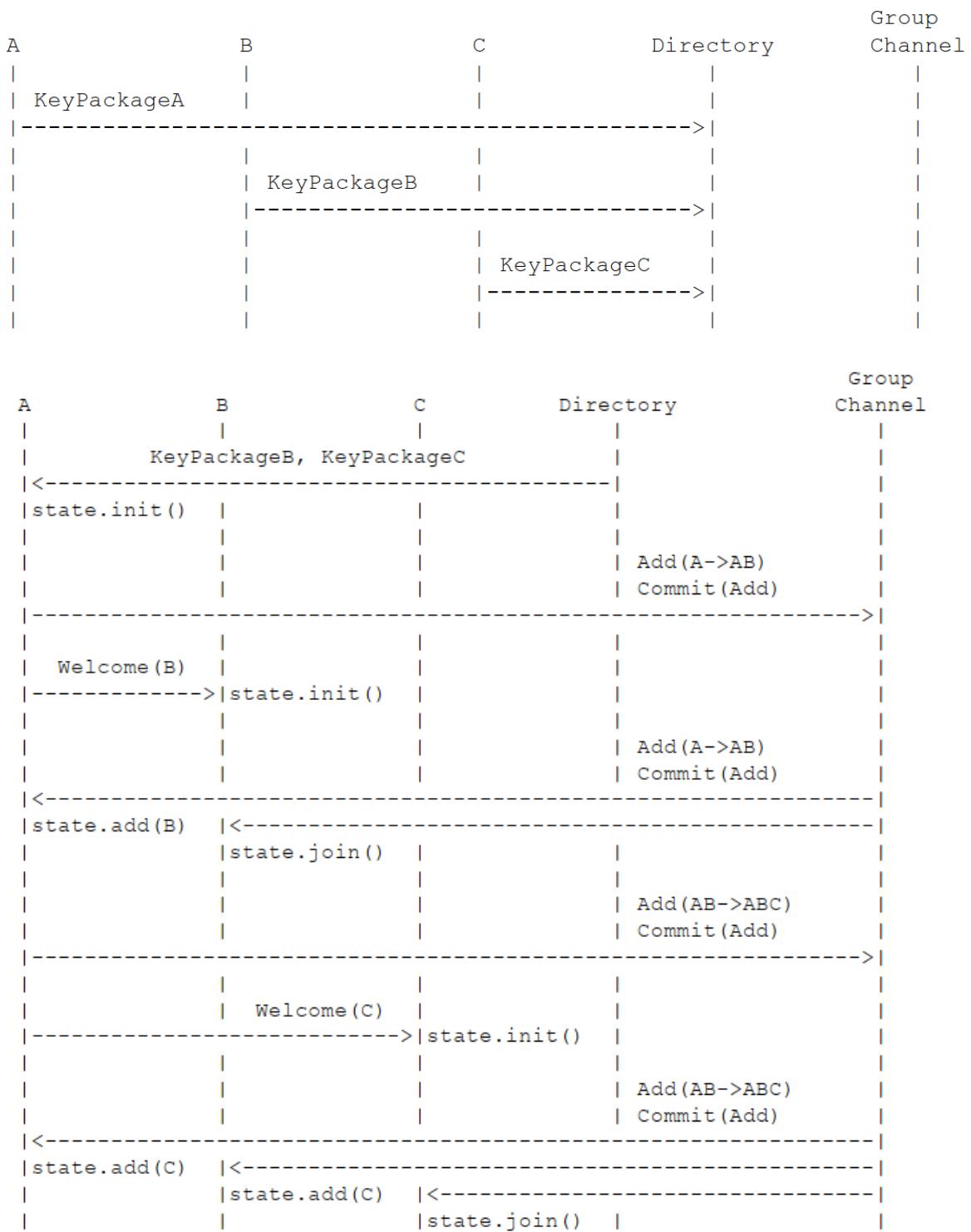
Mỗi thao tác này được "yêu cầu" bằng cách gửi tin nhắn thuộc loại tương ứng (Thêm / Cập nhật / Xóa). Tuy nhiên, trạng thái của nhóm không được thay đổi cho đến khi một thông báo Commit được gửi để cung cấp cho nhóm với thông tin ngẫu nhiên mới. Trong phần này, chúng tôi cho thấy mỗi yêu cầu được cam kết ngay lập tức, nhưng trong các trường hợp triển khai nâng cao hơn, một ứng dụng có thể thu thập một số yêu cầu trước khi thực hiện tất cả các yêu cầu cùng một lúc.

Trước khi khởi tạo một nhóm, client công khai InitKeys (dưới dạng đối tượng KeyPackage) vào một thư mục được cung cấp bởi Dịch vụ nhắn tin.

Khi client A muốn thiết lập một nhóm với B và C [Hình 2.2], trước tiên, nó tải xuống KeyPackages cho B và C. Sau đó, nó khởi tạo trạng thái nhóm chỉ chứa chính nó và sử dụng KeyPackages để tính toán Welcome và Add messages để thêm B và C, trong một trình tự được chọn bởi A. Các Welcome messages được gửi trực tiếp đến các thành viên mới (không cần gửi chúng đến nhóm). Các Add messages được gửi đến nhóm và được xử lý theo trình tự bởi B và C. Tin nhắn nhận được trước khi client tham gia nhóm sẽ bị bỏ qua. Chỉ sau khi A nhận được Add messages từ máy chủ, nó mới cập nhật trạng thái để phản ánh sự bổ sung của họ.

Việc bổ sung các thành viên nhóm sau đó được tiến hành theo cách tương tự. Bất kỳ thành viên nào trong nhóm đều có thể tải xuống KeyPackage của client mới và phát thông báo Add message mà nhóm hiện tại có thể sử dụng để cập nhật trạng thái của họ và Welcome message mà client mới có thể sử dụng để khởi tạo trạng thái của mình.

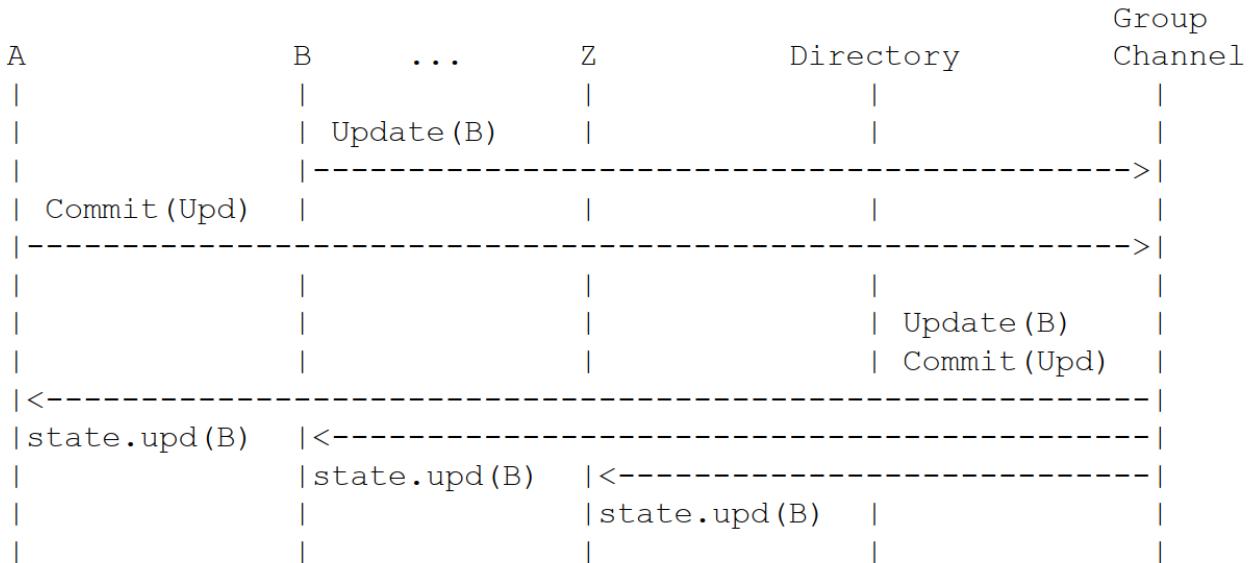
Để thực thi bảo mật về trước và sau thỏa hiệp, mỗi thành viên định kỳ cập nhật bí mật của họ [Hình 2.3]. Bất kỳ thành viên nào cũng có thể cập nhật thông tin này bất cứ lúc nào bằng



Hình 2.2: Quy trình thêm Client vào nhóm

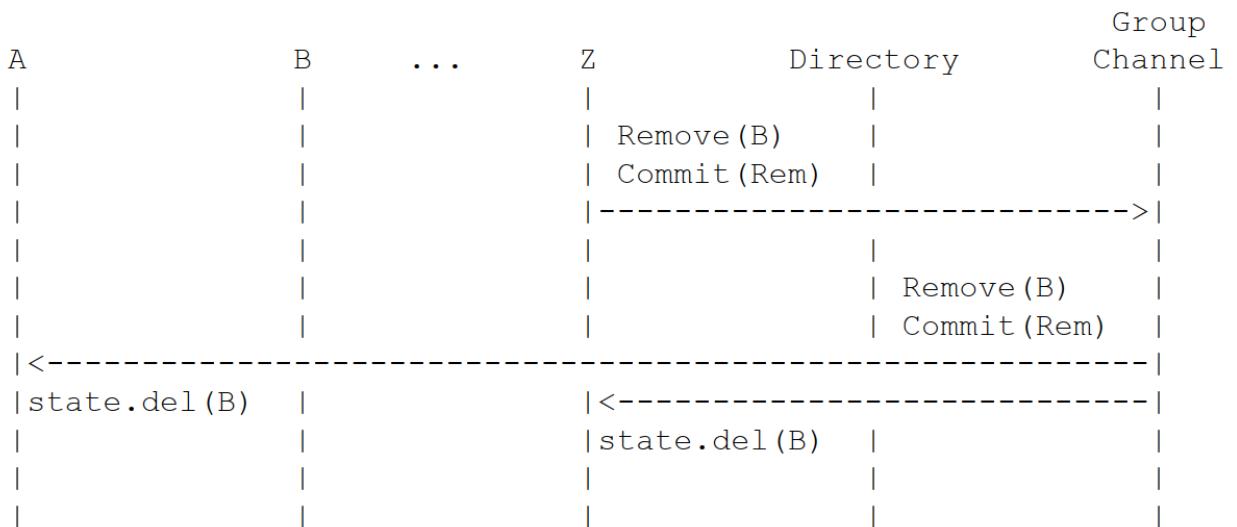
cách tạo KeyPackage mới và gửi Update message theo sau là Commit message. Một khi tất cả các thành viên đã xử lý xong, các bí mật của nhóm sẽ không được biết đến kể cả với kẻ

tấn công đã xâm phạm bí mật trước của người gửi.



Hình 2.3: Quy trình cập nhật key của nhóm

Nó được ứng dụng sử dụng để xác định một chính sách thường xuyên gửi Update message. Chính sách này có thể mạnh như yêu cầu Update + Commit sau mỗi thông báo ứng dụng hoặc yếu hơn, chẳng hạn như mỗi giờ, ngày,... một lần.



Hình 2.4: Quy trình xóa thành viên nhóm

Các thành viên bị xóa khỏi nhóm theo cách tương tự [Hình 2.4]. Bất kỳ thành viên nào trong nhóm đều có thể gửi yêu cầu Remove sau là thông báo Commit, bổ sung thông tin ngẫu

nhiên mới cho trạng thái nhóm mà tất cả mọi người biết, ngoại trừ thành viên bị xóa. Lưu ý rằng điều này không nhất thiết ngụ ý rằng bất kỳ thành viên nào thực sự được phép xóa các thành viên khác; các nhóm có thể thực thi các chính sách kiểm soát truy cập trên cơ chế cơ bản này.

2.5 Ratchet Tree

Giao thức sử dụng thuật toán "ratchet trees" để lấy bí mật chung giữa một nhóm client.

2.5.1 Thuật ngữ tính toán trong cây

Cây bao gồm `_nodes_`. Một nút là một `_leaf_` nếu nó không có con và `_parent_` nếu có; lưu ý rằng tất cả các nút cha trong cây của có chính xác hai con, một `_left_` và một `_right_`. Một nút là `_root_` của cây nếu nó không có cha và `_intermediate_` (trung gian) nếu nó có cả con và bố mẹ. `_Descendants_` của một nút là nút đó, con của nó và con cháu của các con của nó và chúng ta nói một cây `_contains_` một nút nếu nút đó là con của nút gốc. Các nút là `_siblings_` nếu chúng có cùng cha mẹ.

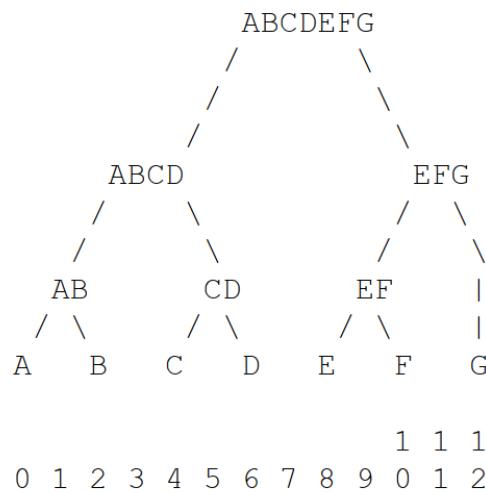
Một `_subtree_` của một cây là cây được đưa ra bởi con cháu của bất kỳ nút nào, là `_head_` của cây con. `_Size_` của cây hoặc cây con là số nút lá mà nó chứa. Đối với một nút cha bất kỳ, `_left subtree_` của nó là cây con lấy nút con bên trái của nó làm gốc (tương ứng là `_right subtree_`).

Tất cả các cây được sử dụng trong giao thức này là cây nhị phân cân bằng trái (left-balanced binary trees). Cây nhị phân là `_full_` (và `_balanced_`) nếu kích thước của nó là lũy thừa của hai và đối với bất kỳ nút cha nào trong cây, các cây con trái và phải của nó có cùng kích thước. Nếu một cây con là `_full_` và nó không phải là một tập hợp con của bất kỳ cây con đầy đủ khác, thì được gọi là `_maximal_`.

Cây nhị phân là `_left-balanced_` nếu với mỗi nút cha, thì nút cha này được cân bằng hoặc cây con bên trái của nút cha đó là cây con đầy đủ lớn nhất có thể được xây dựng từ các lá có trong cây con của nút cha. Đưa ra một danh sách các mục "`n`", có một cây nhị phân cân bằng trái duy nhất với các phần tử này là các lá. Trong một cây cân bằng bên trái như vậy, nút lá "`k-th`" đề cập đến nút lá "`k-th`" trong cây khi đếm từ bên trái, bắt đầu từ 0.

`_direct path_` của một nút gốc là danh sách trống và bất kỳ nút nào là sự kết hợp của nút cha đó cùng với đường dẫn trực tiếp đến nút cha đó. `_copath_` của một nút là anh chị em của nút được nối với danh sách anh chị em của tất cả các nút trong đường dẫn trực tiếp của nó. [Hình 2.5]

Mỗi nút trong cây được gán một `_node index_`, bắt đầu từ 0 và chạy từ trái sang phải. Một nút là một nút lá khi và chỉ khi nó có một chỉ số chẵn. Các chỉ số nút cho các nút trong cây



Hình 2.5: Direct path của C là (CD, ABCD, ABCDEFG).
Copath của C là (D, AB, EFG).

trên như sau: 0 = A ; 1 = AB ; 2 = B ; 3 = ABCD ; 4 = C ; 5 = CD ; 6 = D ; 7 = ABCDEFG ; 8 = E ; 9 = EF ; 10 = F ; 11 = EFG ; 12 = G

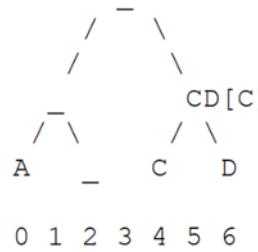
Các lá của cây được lập chỉ mục riêng, sử dụng `_leaf index_`, vì các thông điệp giao thức chỉ cần tham chiếu đến các lá trong cây. Giống như các nút, lá được đánh số từ trái sang phải. Lưu ý rằng với cách đánh số ở trên, một nút là một nút lá khi và chỉ khi nó có một chỉ số nút chẵn và một nút lá có chỉ số của nút là một nửa chỉ số nút cha của nó. Các chỉ số lá trong cây trên như sau: 0 = A ; 1 = B ; 2 = C ; 3 = D ; 4 = E ; 5 = F ; 6 = G

Một nút lá `_unmerge_` khi nó được thêm vào lần đầu tiên, bởi vì quá trình thêm nút lá không cho phép nó truy cập vào tất cả các nút phía trên nó trong cây. Các lá được `_merged_` khi chúng nhận được các Khóa bí mật của các nút.

Một nút trong cây cũng có thể là `_blank_`, chỉ ra rằng không có giá trị nào hiện diện tại nút đó. `_Resolution_` của một nút là một danh sách có thứ tự các nút không trống (not blank) bao gồm tất cả các node con cháu không trống của nút đó: [Hình 2.6]

- `_resolution_` của nút không trống bao gồm chính nút đó, theo sau là danh sách các lá `_unmerge_` của nó, nếu có.
- `_resolution_` của nút lá trống là danh sách trống.
- `_resolution_` của nút trung gian (`_intermediate_`) trống là kết quả của `_resolution_` của nút con bên trái của nó với độ phân giải của nút con bên phải của nó.

Mỗi nút, bất kể nút đó là trống hay không, đều có `_hash_` tương ứng chứa các thông tin tóm



Hình 2.6: Resolution của node 5 là danh sách [CD, C]

Resolution của node 2 là danh sách trống []

Resolution của node 3 là danh sách [A, CD, C]

tắt của nút con bên dưới nút đó

2.5.2 Ratchet Tree Nodes

Một phiên bản cụ thể của ratchet tree dựa trên các nguyên hàm mã hóa sau, được xác định bởi mật mã đang sử dụng:

- Một mật mã HPKE, chỉ định Cơ chế đóng gói khóa (Key Encapsulation Mechanism - KEM), sơ đồ mã hóa AEAD và hàm băm.
- Hàm Derive-Key-Pair tạo ra cặp khóa bất đối xứng cho KEM được chỉ định từ một bí mật đối xứng (a symmetric secret).

Mỗi nút trong cây ratchet chứa tối đa 5 giá trị:

- Khóa bí mật
- Khóa công khai
- Một danh sách theo thứ tự các chỉ số lá cho những nút _unmerge_.
- Một thông tin xác thực (chỉ dành cho các nút lá).
- Mã hash của nút cha, kể từ lần cuối cùng nút đó được thay đổi.

2.6 Basic background on TreeKEM

Mỗi người dùng biết tất cả các bí mật nút trên đường dẫn đến lá của họ, cũng như tất cả các khóa công khai trong cây, nhưng họ không thể tính toán các bí mật nút khác trong cây.

Một hàm Derive-Key-Pair(DKP) ánh xạ các chuỗi nhị phân thành các cặp khóa bất đối xứng (private key; public key).

- (private key; public key) = DKP(node secret)

Hàm Key Derivation Function(KDF) ánh xạ các cặp nhị phân của chuỗi nhị phân thành chuỗi

nhi phân, mà chúng ta coi là hàm băm hai đầu vào chống va chạm(Collision resistance)

- Collision resistance là một thuộc tính của hàm băm mật mã: hàm băm H có khả năng chống va chạm nếu khó tìm thấy hai đầu vào băm cho cùng một đầu ra; nghĩa là, hai đầu vào a và b sao cho $H(a) = H(b)$ và $a \neq b$
- Sử dụng KDF để tạo 1 secret mới từ secret cũ: $(secret\ 2) = KDF((secret\ 1);\ "next")$

Mỗi nút có 3 giá trị:

- Node secret
- Public key
- Secret key

2.6.1 Key Update

2.6.2 Init and Application Secrets

2.6.3 Adding and Removing Users

Chương 3

PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1 Giới thiệu hệ thống

Khóa luận hướng đến việc áp dụng công nghệ blockchain để giải quyết các vấn đề về tính minh bạch, công khai và tin cậy của ứng dụng gây quỹ cộng đồng theo mô hình truyền thống đã đề cập ở mục 1.1, 2.2. Nhóm tác giả sử dụng blockchain cho việc lưu trữ các thông tin về tài chính, các giao dịch của người dùng trong hệ thống nhằm hạn chế việc lưu các thông tin này trên bất kì một bên thứ ba nào. Hơn thế nữa, hệ thống sử dụng hợp đồng thông minh cho các lệnh liên quan tài chính trong hệ thống. Các lệnh này được thực hiện một cách tự động và chính xác, tránh sự can thiệp hay tác động từ yếu tố con người vào hệ thống. Với hợp đồng thông minh, nhóm tác giả cũng bổ sung các tính năng như tự động hoàn tiền khi mục tiêu gây quỹ chiến dịch không thành công; bỏ phiếu để giải ngân, tăng quyền hạn cho người đóng góp chiến dịch.

Bên cạnh đó, hệ thống của nhóm tác giả cũng sử dụng một phần dữ liệu được lưu trữ tập trung nhằm cải thiện tốc độ đọc ghi với các dữ liệu không cần độ tin cậy cao.

3.2 Quy trình gây quỹ cộng đồng

Đây là quy trình gây quỹ cộng đồng trong hệ thống của nhóm tác giả.

3.2.1 Các đối tượng trong quy trình

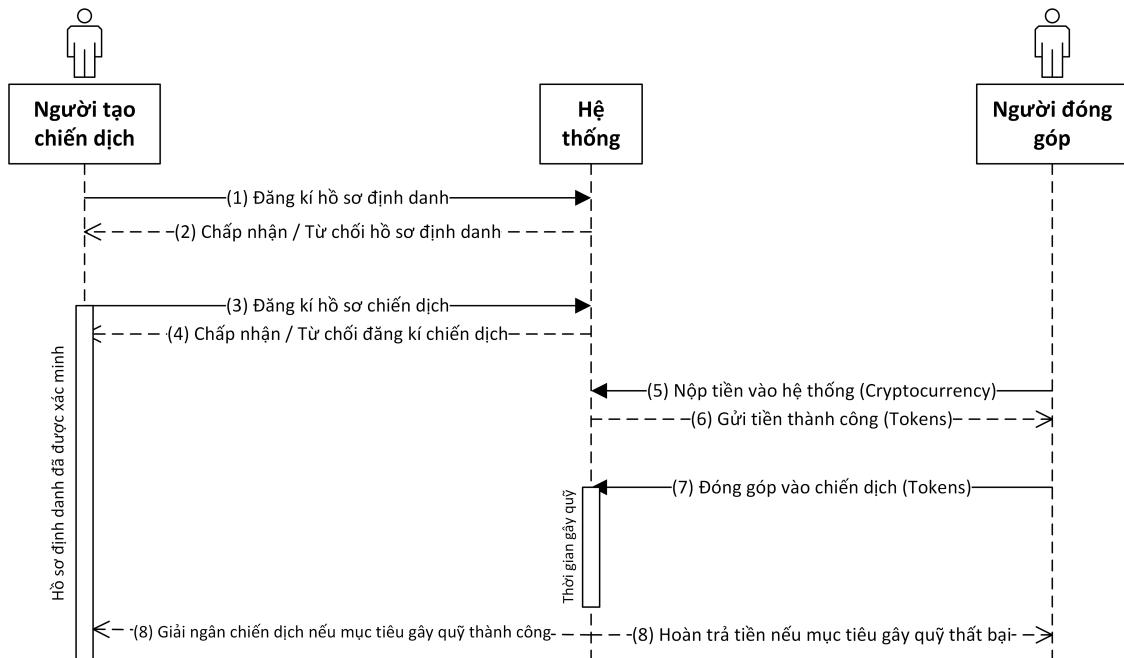
Trong quy trình gây quỹ, có các đối tượng sau:

- **Người tạo chiến dịch:** là những cá nhân / tổ chức có nhu cầu gây quỹ vì mục đích từ thiện, hướng tới cộng đồng.

- **Hệ thống:** là hệ thống gây quỹ trung gian, đứng giữa người tạo chiến dịch và người đóng góp.
- **Người đóng góp:** là người ủng hộ đóng góp tiền cho chiến dịch gây quỹ.

3.2.2 Sơ đồ quy trình gây quỹ

Sơ đồ quy trình gây quỹ được thể hiện ở hình 3.1.



Hình 3.1: Sơ đồ quy trình gây quỹ cộng đồng

Quy trình này được diễn giải như sau:

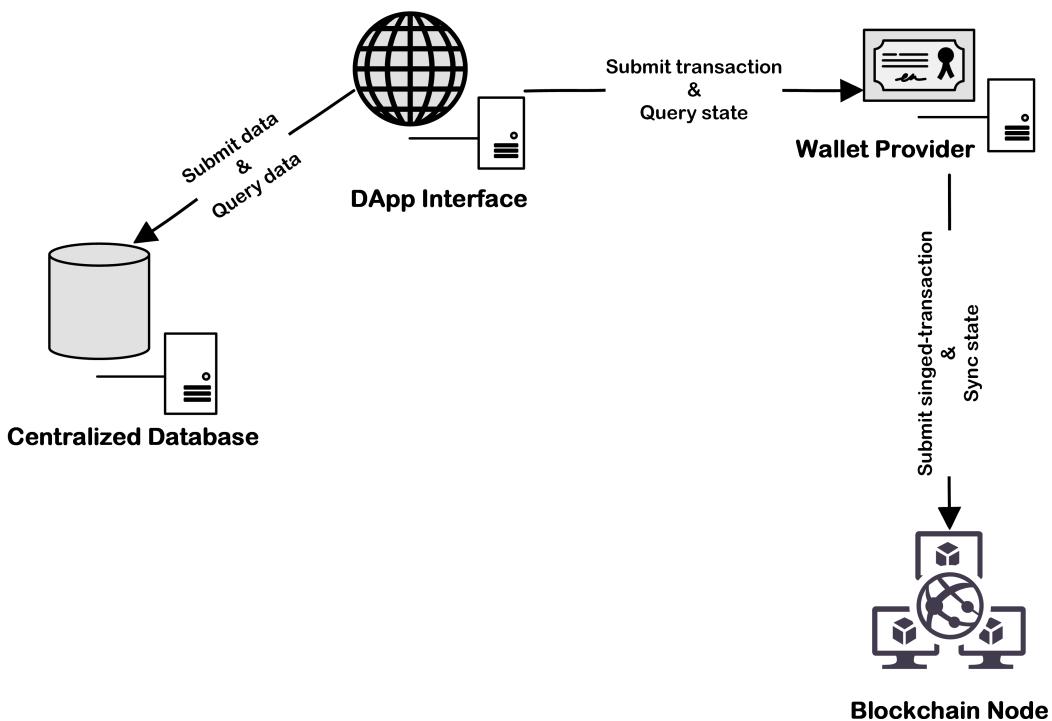
- (1) Người tạo chiến dịch sẽ tiến hành tạo lập hồ sơ định danh bao gồm thông tin cá nhân cơ bản và thông tin chứng minh định danh.
- (2) Hệ thống (nhân viên xác minh) tiến hành xác minh hồ sơ định danh và chấp nhận hay từ chối hồ sơ. Nếu hồ sơ định danh được chấp nhận thì hồ sơ đó được phép gọi lệnh tạo chiến dịch, ngược lại thì không.
- (3) Người tạo chiến dịch tiếp tục tạo lập hồ sơ chiến dịch gây quỹ nếu hồ sơ định danh được chấp nhận.
- (4) Hệ thống (nhân viên xác minh) tiến hành xác minh hồ sơ chiến dịch và chấp nhận hay từ chối hồ sơ. Hồ sơ được chấp nhận sẽ được công khai lên hệ thống và cho phép người đóng góp ủng hộ tiền. Ngược lại thì không.
- (5) Người đóng góp muốn ủng hộ tiền cho một chiến dịch thì cần sử dụng cryptocurrency (đồng tiền mã hóa) gửi vào hệ thống (lúc này là hợp đồng thông minh) để sử dụng các

chức năng trong hệ thống.

- (6) Sau khi người đóng góp gửi tiền vào hệ thống, hệ thống sẽ lưu số tiền người gửi vào dưới dạng một giá trị được gọi là token. Người đóng góp sử dụng token này trong các giao dịch nội bộ của hệ thống. Token này có thể được đổi ngược lại sang đồng cryptocurrency với giá tương ứng
- (7) Người đóng góp ủng hộ tiền cho một chiến dịch (chiến dịch đã được xác minh) bằng một lượng token mà người đóng góp mong muốn và đang có.
- (8) Mỗi chiến dịch sẽ có một khoảng thời gian để kêu gọi đóng góp, và một mục tiêu là số lượng token cần đạt được. Khi hết thời gian kêu gọi đóng góp, nếu chiến dịch hoàn thành mục tiêu thì sẽ tiến hành cho người tạo chiến dịch giải ngân. Ngược lại, lượng token đã đóng góp sẽ được hoàn lại cho người đóng góp.

3.3 Kiến trúc hệ thống

Kiến trúc hệ thống được thể hiện ở hình 3.2.



Hình 3.2: Sơ đồ kiến trúc hệ thống

Các thành phần và vai trò của từng thành phần trong hệ thống mà nhóm tác giả đề xuất như sau:

- **DApp Interface** - là một giao diện ứng dụng phi tập trung, nơi người dùng sẽ tương tác

trực tiếp. *DApp Interface* sẽ tạo ra các transaction để gọi các hàm có trong hợp đồng thông minh và chuyển các transaction này tới Wallet Provider thực hiện công đoạn tiếp theo. Và đây cũng là nơi tương tác với cơ sở dữ liệu tập trung trong mô hình. Thực chất *DApp Interface* chỉ là một giao diện front-end tĩnh chạy ở phía người dùng.

- **Wallet Provider** - là ứng dụng phi tập trung có nhiệm vụ xác nhận và kí transaction (*sign transaction*) do *DApp Interface* gửi tới trong mô hình, sau đó thực hiện gửi các transaction đã được kí (*signed-transaction*) đến mạng lưới blockchain. Wallet cũng làm nhiệm vụ lưu trữ thông tin về khóa bí mật của người dùng.
- **Blokchain Node** - là một nút trong mạng blockchain mà *Wallet Provider* tương tác để lưu trữ các dữ liệu phi tập trung, các dữ liệu phi tập trung trong trường hợp này là các mã hợp đồng thông tin và các giá trị trong hợp đồng.
- **Centralized Database** - là một cơ sở dữ liệu tập trung, được dùng để lưu trữ một số thông tin như mô tả chiến dịch.

3.4 Các chức năng

3.4.1 Tổng quan các chức năng

Tổng quan các chức năng trong hệ thống được thể hiện ở hình 3.3.

Với sơ đồ được thể hiện có các đối tượng sau:

- **Người tạo chiến dịch** - là người tạo chiến dịch gây quỹ.
- **Người đóng góp** - là người đóng góp, ủng hộ tiền cho chiến dịch gây quỹ.
- **Nhân viên xác minh** - là người xác minh cho hồ sơ định danh và hồ sơ gây quỹ, là nhân viên trong hệ thống hoặc tình nguyện viên của hệ thống.
- **Nhân viên vận hành** - là người vận hành hệ thống hay người triển khai các hợp đồng thông minh.

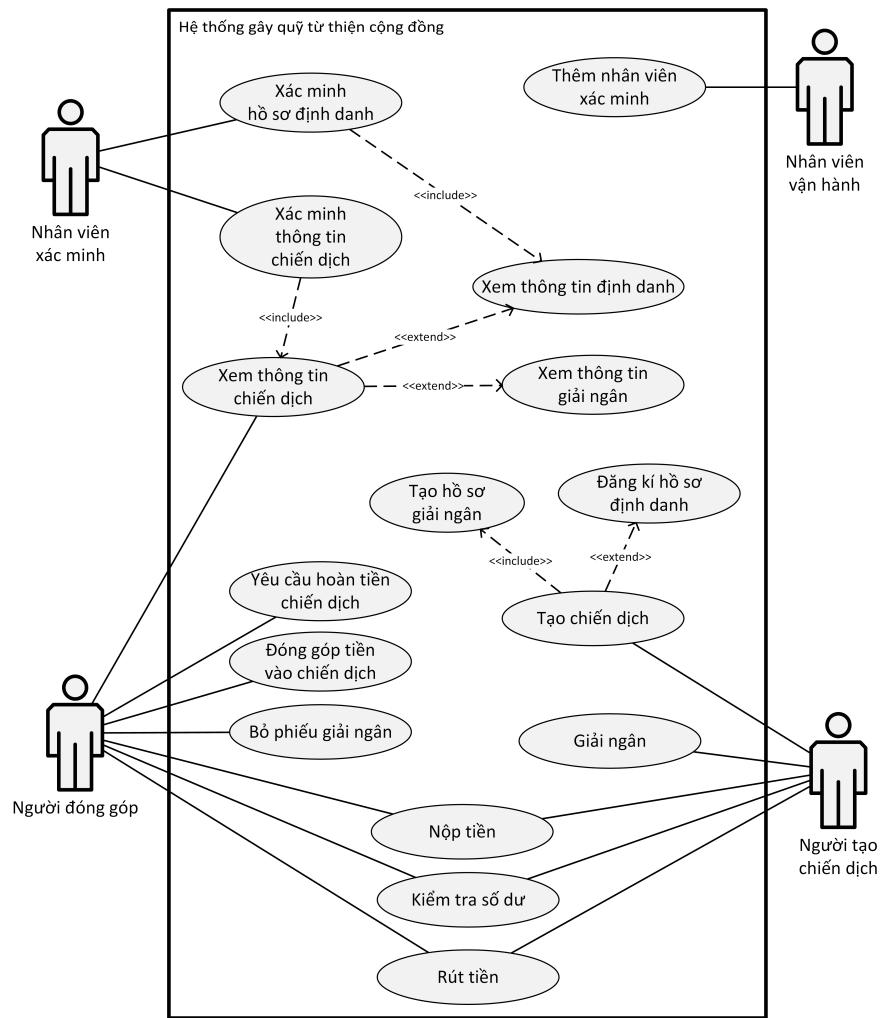
Nhóm tác giả cũng thực hiện phân rã các chức năng và được sơ đồ như ở hình 3.4.

Tiếp theo là sơ đồ ở hình 3.5 thể hiện tổng quan về các đối tượng và luồng tương tác dữ liệu giữa các đối tượng. Đối tượng “**Hệ thống**” trong sơ đồ được nhóm tác giả đặc tả một cách khái quát cho toàn thể hệ thống.

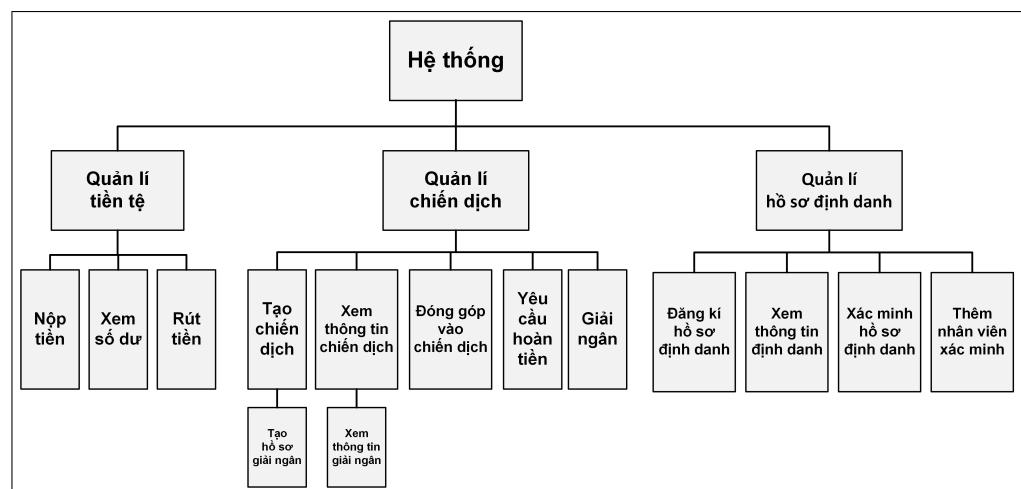
3.4.2 Chức năng nộp tiền và rút tiền

3.4.2.1 Mục tiêu

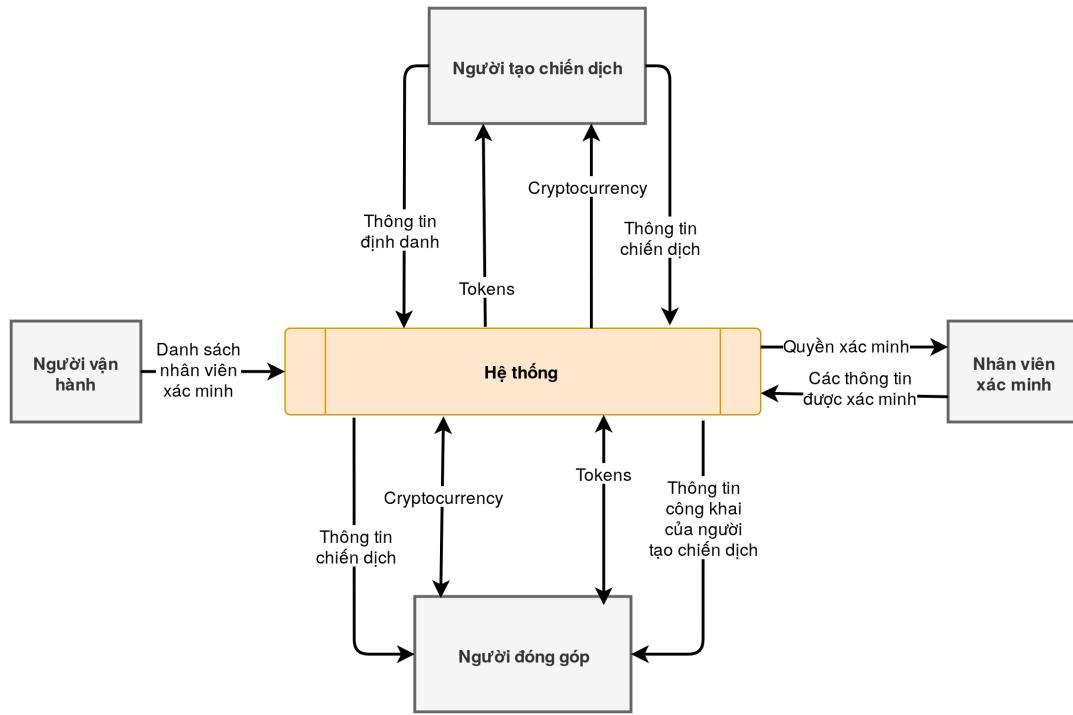
Trong mô hình hệ thống mà nhóm tác giả đề xuất, người đóng góp sẽ ủng hộ cho chiến dịch gây quỹ từ thiện thông qua tiền mã hóa. Trong thời gian kêu gọi quỹ, người tạo chiến dịch



Hình 3.3: Sơ đồ tổng quan các chức năng trong hệ thống



Hình 3.4: Sơ đồ phân rã các chức năng trong hệ thống



Hình 3.5: Sơ đồ tổng quan các đối tượng và luồng tương tác dữ liệu

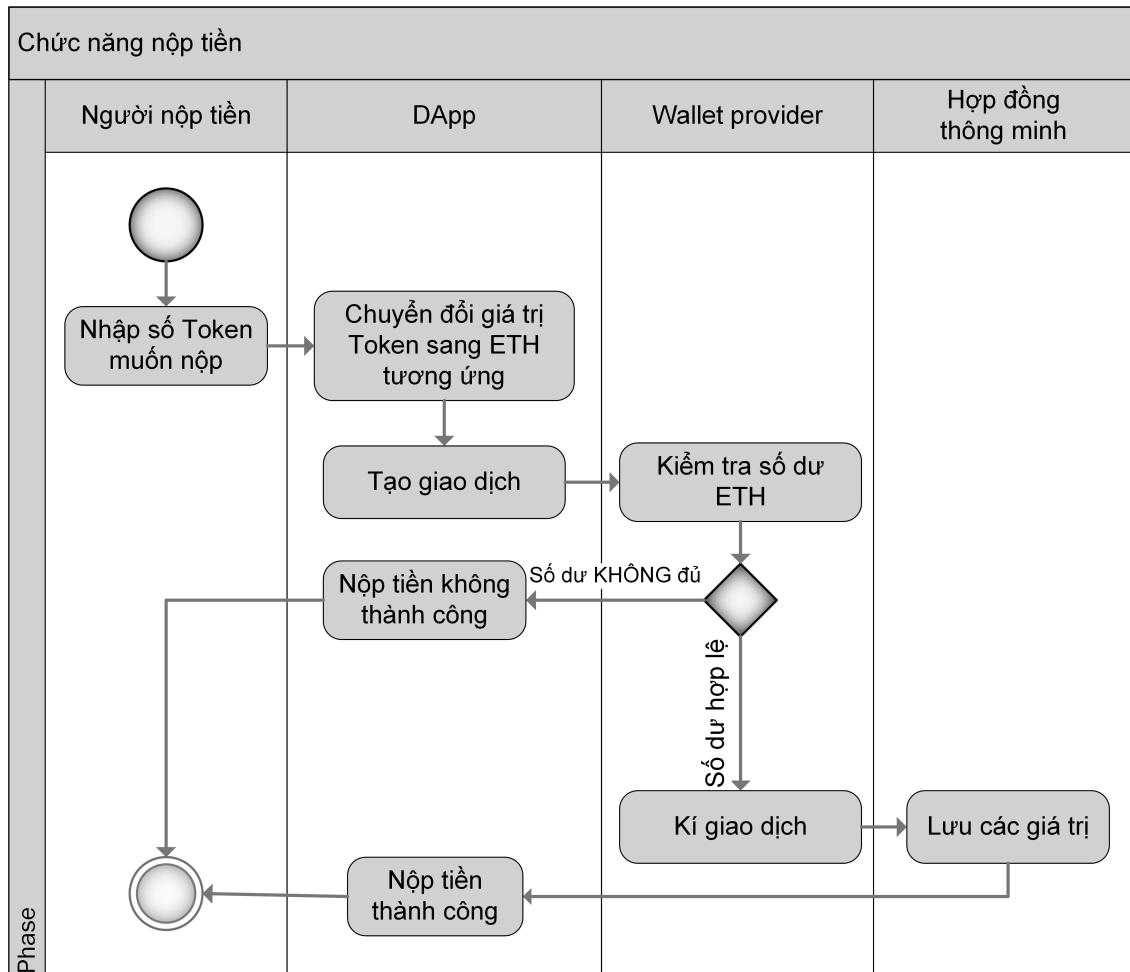
chưa nhận được bất kì lượng tiền nào, chỉ khi nào chiến dịch kêu gọi đạt được mục tiêu thì người tạo chiến dịch mới thực sự nhận được tiền. Để đạt được yêu cầu như vậy thì trong thời gian kêu gọi quỹ, người đóng góp chỉ thật sự ủng hộ trên danh nghĩa là đồng ý đóng góp khoản tiền nào đó chứ không chuyển trực tiếp cho người tạo chiến dịch. Việc ủng hộ trên danh nghĩa này phải được đảm bảo rằng sau khi chiến dịch kết thúc, người tạo chiến dịch sẽ nhận được đúng số tiền ủng hộ. Do đó, trong trường hợp này sẽ cần xuất hiện một bên thứ ba, làm nhiệm vụ giữ tiền và trao đổi tiền giữa hai bên. Như đã được giới thiệu từ trước, hợp đồng thông minh đảm bảo việc thực thi các luồng xử lí một cách tự động mà không có sự can thiệp từ yếu tố con người làm sai lệch kết quả. Nên sử dụng hợp đồng để lưu giữ tiền cam kết giữa hai bên và xử lí các điều kiện để giải ngân cho chiến dịch là hợp lý. Vì vậy cần có chức năng gửi tiền và rút tiền từ hợp đồng thông minh với mục tiêu sau:

- Người đóng góp sẽ gửi tiền mã hóa vào hợp đồng minh và nhận lại một giá trị lưu trữ cho số tiền gửi vào. Giá trị này được gọi là **token**.
- Token được sử dụng cho các hoạt động trong hệ thống, bao gồm đóng góp vào chiến dịch gây quỹ.
- Token cũng được quy đổi ngược lại đồng tiền mã hóa theo một tỉ lệ tương ứng nhất định. Đây là chức năng rút tiền trong hệ thống.
- Tỉ lệ quy đổi sẽ do người vận hành hệ thống quyết định.

- Giá trị token của người dùng đã gửi vào luôn không thay đổi trong suốt quá trình tham gia hệ thống. Ngoại trừ hoạt động rút tiền và đóng góp cho chiến dịch.

3.4.2.2 Cách thức hoạt động

Sơ đồ hoạt động chức năng nộp tiền được thể hiện ở hình 3.6.



Hình 3.6: Sơ đồ hoạt động chức năng nộp tiền

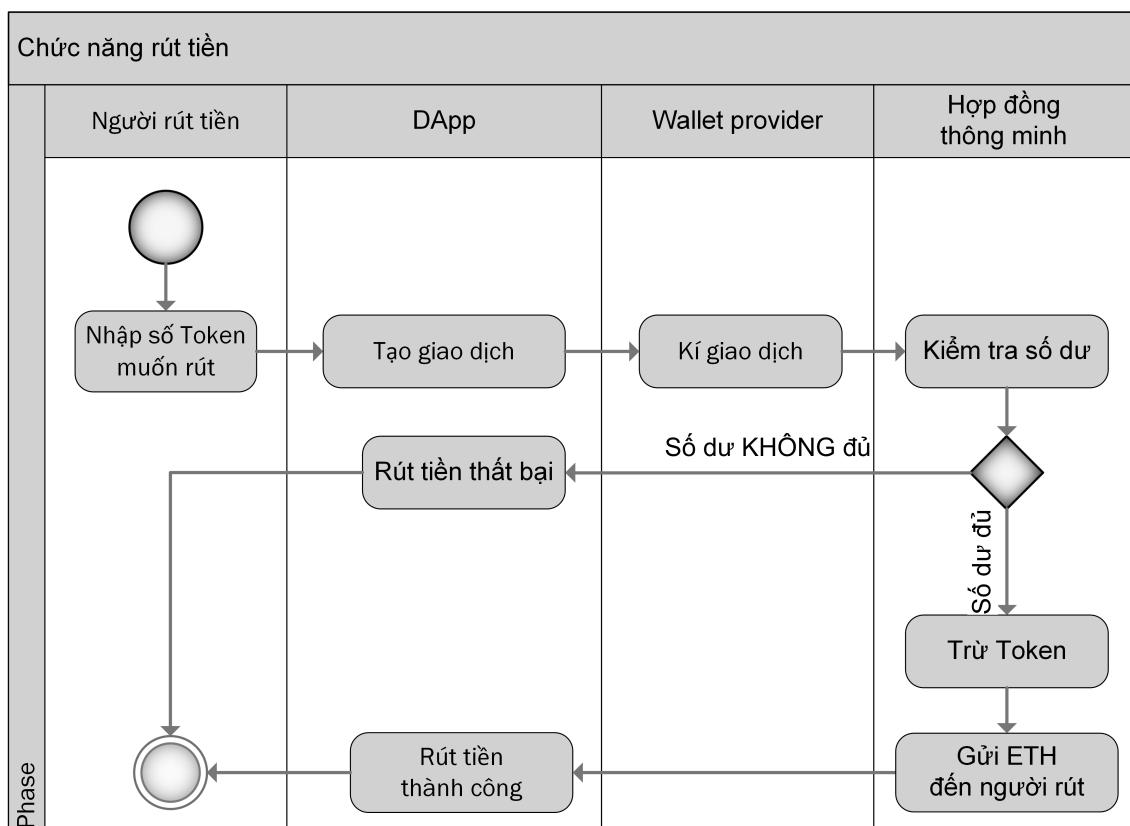
Tiến trình của hoạt động nộp tiền được diễn ra như sau:

- (1) Người dùng nhập vào lượng token muốn gửi vào hệ thống.
- (2) Trên giao diện DApp thực hiện tính toán giá trị token và quy đổi sang lượng ether tương ứng.
- (3) DApp tiến hành tạo transaction và gửi đến Wallet provider.
- (4) Wallet provider tiến hành kiểm tra số dư ether của người dùng.
- (5) Nếu số dư đủ để tiến hành giao dịch thì Wallet provider bắt đầu kí transaction và gửi

tới một nút trong mạng blockchain. Ngược lại, hiển thị thông báo lỗi cho người dùng trên giao diện DApp.

- (6) Một nút trong blockchain nhận được transaction từ Wallet provider, lúc này hàm gửi tiền trong hợp đồng thông minh được thực thi. Hợp đồng thông minh tiến hành lưu trữ giá trị token.
- (7) Hiển thị thông báo gửi tiền thành công và kết thúc tiến trình.

Sơ đồ hoạt động chức năng rút tiền được thể hiện ở hình 3.7.



Hình 3.7: Sơ đồ hoạt động chức năng rút tiền

Tiến trình của hoạt động rút tiền được diễn ra như sau:

- (1) Người dùng nhập vào lượng token muốn rút.
- (2) DApp tiến hành tạo transaction và gửi đến Wallet provider.
- (3) Wallet provider tiến hành kí transaction và gửi tới một nút trong mạng blockchain.
- (4) Một nút trong blockchain nhận được transaction từ Wallet provider, lúc này hàm rút tiền trong hợp đồng thông minh được thực thi. Hợp đồng thông minh tiến hành kiểm tra giá trị token của người dùng. Nếu số dư token đủ thì tiến hành giảm trừ số token

tương ứng, sau đó gửi lượng tiền mã hóa ether tương ứng cho người dùng, sau đó hiển thị thông báo rút tiền thành công và kết thúc tiến trình. Ngược lại, thông báo lỗi cho người dùng trên giao diện DApp.

3.4.3 Chức năng quản lý định danh

3.4.3.1 Mục tiêu

Do trong blockchain mỗi người dùng sẽ được xác định bởi các address (địa chỉ), các địa chỉ này hoàn toàn tách biệt với danh tính của người dùng, tức nó không bao gồm danh tính hay bất cứ thông tin nào như địa chỉ IP, định vị, Do đó có thể nói mỗi người dùng trên mạng blockchain là ẩn danh **henry2018blockchain**. Để tăng tính tin cậy cho chiến dịch gây quỹ thì cần thiết phải gắn mỗi địa chỉ người dùng cho một hồ sơ định danh, vì vậy một địa chỉ người dùng muốn đăng ký tạo chiến dịch thì bắt buộc địa chỉ đó đã có hồ sơ định danh và hồ sơ định danh đó phải được xác minh. Việc tạo lập hồ sơ định danh chỉ bắt buộc với địa chỉ người dùng nào muốn tạo chiến dịch, còn đối với người đóng góp vào chiến dịch thì không bắt buộc.

Hồ sơ định danh có 2 loại thông tin cơ bản là:

- **Thông tin công khai:** là những thông tin cơ bản của người tạo chiến dịch như họ tên, địa chỉ, ngày sinh. Việc công khai thông tin là bắt buộc đối với người tạo chiến dịch.
- **Thông tin cá nhân nhạy cảm:** là các thông tin cá nhân bí mật, được dùng để chứng minh cho các thông tin được công khai. Do đó cần lưu trữ thông tin cá nhân nhạy cảm một cách bí mật và toàn vẹn.

Yêu cầu về chia sẻ thông tin cá nhân giữa người dùng và người xác minh phải đảm bảo được các yếu tố:

1. Chỉ có người dùng và người xác minh mới có thể đọc được thông tin.
2. Việc xác minh cho một hồ sơ được minh bạch. Tức biết ai là người đã xác minh cho hồ sơ, và vào thời gian nào.

3.4.3.2 Cách hoạt động

Nhóm tác giả chia làm 2 tiến trình hoạt động cho chức năng này:

- Tạo lập và lưu trữ hồ sơ định danh.
- Chia sẻ thông tin hồ sơ định danh.

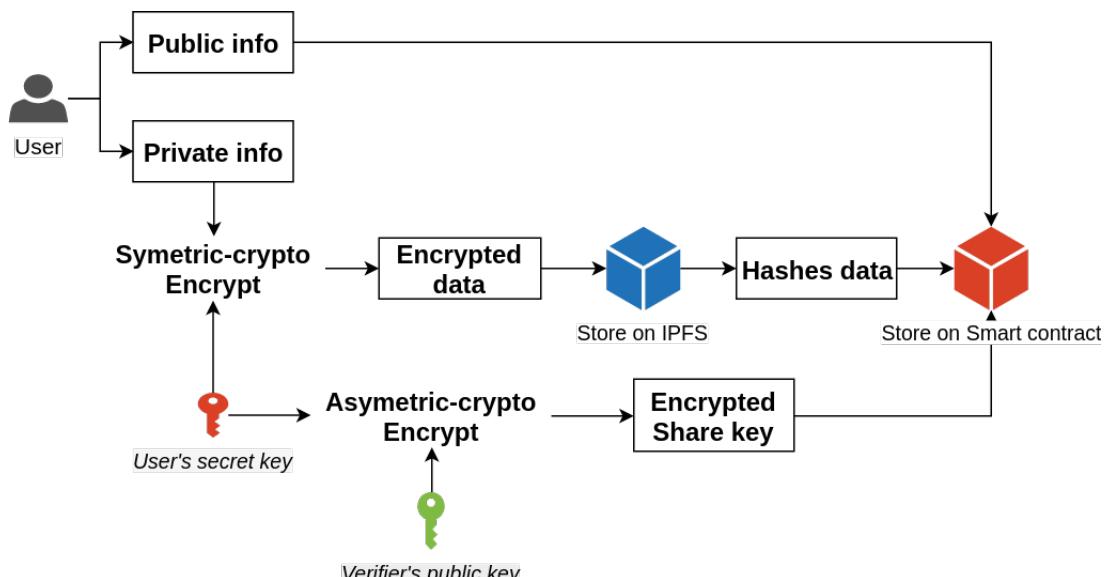
Các đối tượng trong chức năng định danh bao gồm:

- **Người tạo lập hồ sơ:** là người tạo hồ sơ định danh, hay người tạo chiến dịch gây quỹ.

- **Nhân viên xác minh:** người xác minh cho một hồ sơ định danh. Có thể là nhân viên trong hệ thống, tình nguyện viên, cơ quan chuyên trách có nghiệp vụ về xác minh.
- **Nhân viên vận hành:** người quản lý danh sách các nhân viên xác minh. Nhân viên vận hành còn là người sẽ triển khai hợp đồng thông minh lên blockchain.

Cách thức tạo lập và lưu trữ hồ sơ định danh được thể hiện ở hình 3.8. Cụ thể:

- Người tạo lập hồ sơ định danh tiến hành nhập thông tin định danh.
- Người tạo lập hồ sơ nhập một chìa khóa bảo vệ hồ sơ định danh, được gọi là **SecretKey**. SecretKey được dùng cho 2 mục đích:
 - (i) Làm khóa (key) cho thuật toán AES dùng để mã hóa các thông tin nhạy cảm của người dùng trước khi lưu trữ.
 - (ii) SecretKey sẽ được mã hóa bằng thuật toán RSA bởi khóa công khai của nhân viên xác minh, sau đó chuỗi mã hóa sẽ được lưu trữ trên blockchain.
- Người tạo lập chọn khóa công khai của nhân viên xác minh (chọn một nhân viên xác minh trong danh sách các nhân viên xác minh), khóa này được dùng để mã hóa SecretKey của người tạo lập hồ sơ.
- Thông tin công khai và khóa bí mật đã mã hóa sẽ được lưu trữ trên hợp đồng thông minh trong blockchain. Thông tin bí mật đã mã hóa sẽ được lưu trữ trên một mạng lưu trữ phi tập trung (ở đây nhóm tác giả đề xuất IPFS).

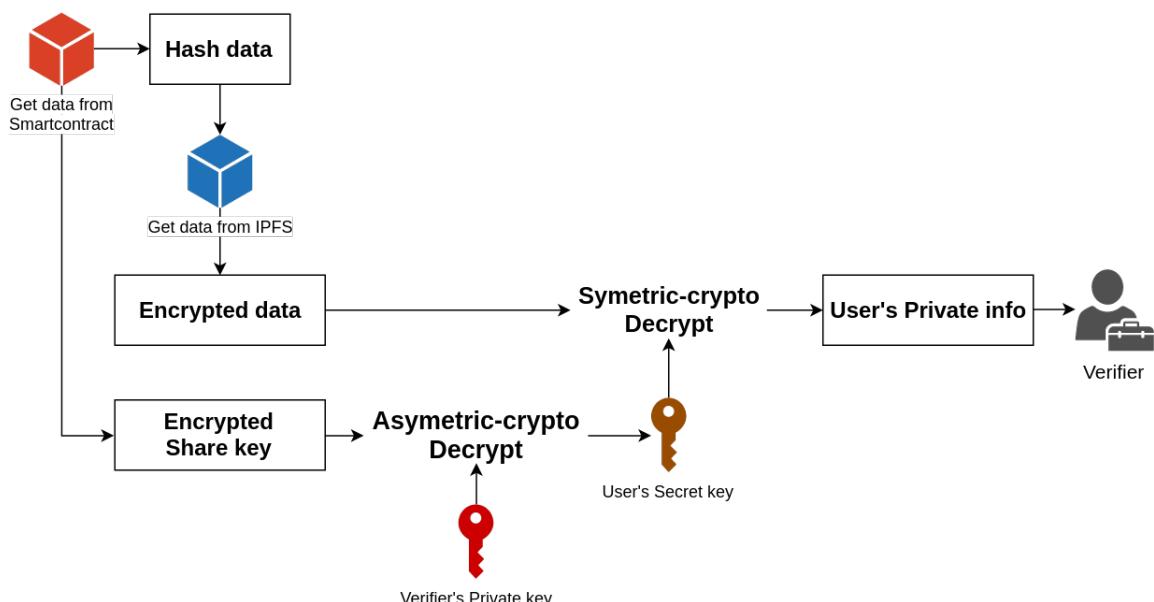


Hình 3.8: Sơ đồ cách thức lưu trữ hồ sơ định danh

Việc chia sẻ hồ sơ định danh là quá trình chia sẻ các thông tin bí mật giữa nhân viên xác

minh và người tạo lập hồ sơ phục vụ cho quy trình xác minh hồ sơ. Cách thức chia sẻ hồ sơ định danh được thể hiện trong hình 3.9:

- Nhân viên xác minh chọn một hồ sơ định danh. Danh sách hồ sơ mà nhân viên xác minh chọn để duyệt là các hồ sơ mà người tạo hồ sơ đã chọn khóa công khai tương ứng với nhân viên xác minh đó.
- Nhân viên xác minh sẽ nhập khóa bí mật (*private key*) để giải mã *SecretKey* của người tạo lập hồ sơ đã được mã hoá bởi khóa công khai của nhân viên xác minh trước đó.
- Sau khi có được *SecretKey* của người dùng, nhân viên xác minh sẽ tiến hành giải mã thông tin hồ sơ bí mật.
- Nhân viên xác minh dựa vào thông tin bí mật đã giải mã và tiến hành xác minh.



Hình 3.9: Sơ đồ cách thức chia sẻ thông tin định danh

3.4.4 Tạo lập và lưu trữ chiến dịch gây quỹ

3.4.4.1 Mục tiêu

Mục tiêu đối với chức năng tạo lập chiến dịch:

- Tạo điều kiện tốt nhất và thuận lợi cho người tạo chiến dịch có thể đăng ký được chiến dịch gây quỹ.
- Người tạo lập chiến dịch cần đăng ký hồ sơ định danh trước khi gọi lệnh đăng ký chiến dịch gây quỹ.

Đối với việc lưu trữ chiến dịch gây quỹ:

- Các thông tin liên quan tài chính thì đặt ưu tiên lưu trữ trên blockchain, đảm bảo tính toàn vẹn, công khai và minh bạch.
- Các dữ liệu không tài chính thì có thể lưu trữ trên cơ sở dữ liệu tập trung, ưu tiên tốc độ đọc dữ liệu.

3.4.4.2 Cách thức hoạt động

Sơ đồ hoạt động của tiến trình tạo chiến dịch được thể hiện ở hình 3.10. Tiến trình cụ thể như sau:

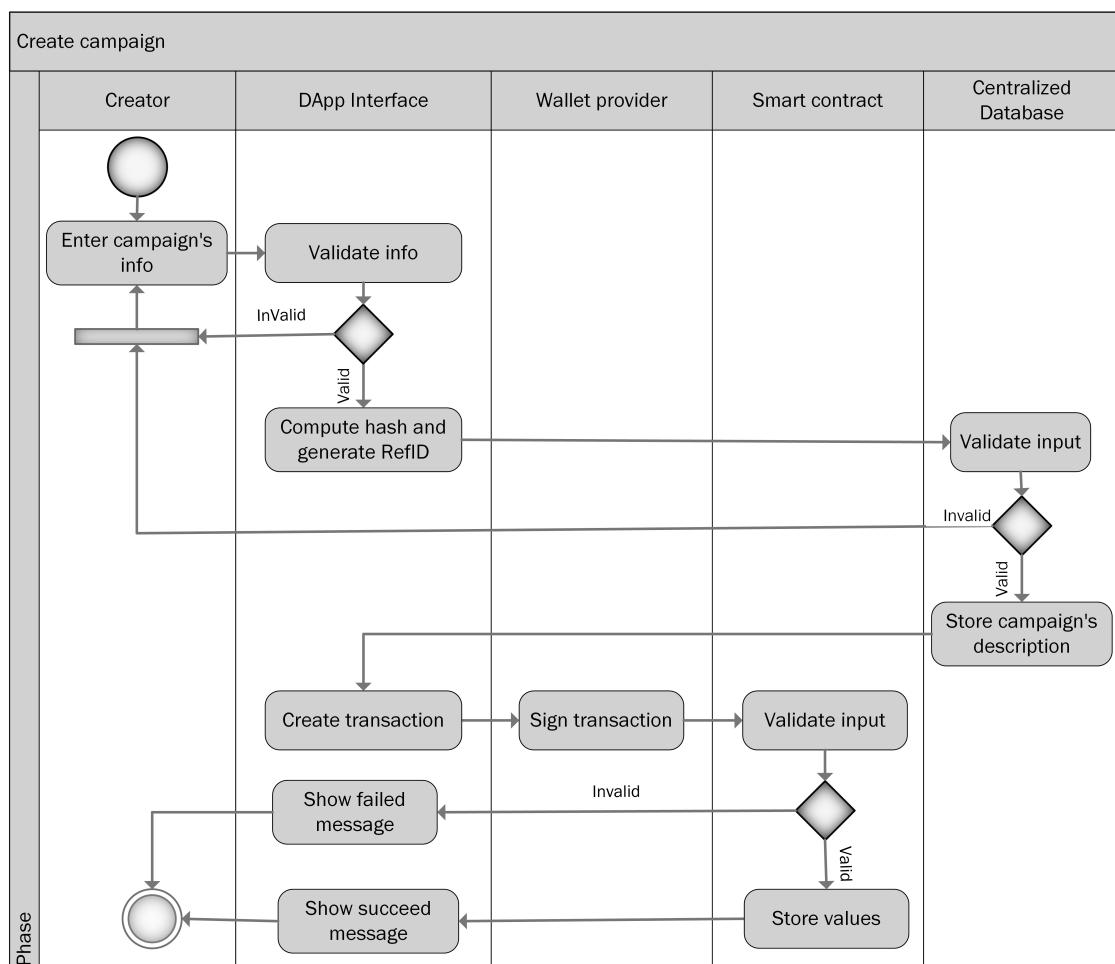
- Đầu tiên người tạo chiến dịch (creator) sẽ tiến hành nhập vào các thông tin của chiến dịch trên DApp Interface. Các thông tin này bao gồm:
 - Tên chiến dịch gây quỹ.
 - Mô tả ngắn gọn về chiến dịch gây quỹ.
 - Mô tả đầy đủ về chiến dịch.
 - Ảnh (video) về chiến dịch.
 - Mục tiêu gây quỹ.
 - Thời gian gây quỹ.
 - Hồ sơ giải ngân: số giai đoạn giải ngân, số tiền cho từng giai đoạn, phương thức giải ngân.
- DApp Interface tiến hành kiểm tra thông tin mà người tạo đã nhập, nếu thông tin hợp lệ, dữ liệu được xử lý ở tiến trình tiếp theo. Ngược lại, người tạo phải nhập lại thông tin.
- Dữ liệu sau khi được kiểm tra ở DApp, sẽ được tính toán mã băm (hash) và tính toán một giá trị gọi là **RefID**, giá trị này được định nghĩa như sau:

```
RefID = hash(Campaign's name + NOW() + RANDOMIZE())
```

Trong đó: *NOW()* là thời gian hiện dưới dạng số (timestamp); *RANDOMIZE()* là một số ngẫu nhiên.

- RefID cùng với các thông tin chiến dịch như: tên chiến dịch, mô tả ngắn gọn về chiến dịch, mô tả đầy đủ về chiến dịch, ảnh (video) về chiến dịch. Sẽ được gửi tới một cơ sở dữ liệu tập trung của hệ thống (Centralized database). Lúc này cơ sở dữ liệu tập trung này sẽ kiểm tra lại dữ liệu được gửi tới lần nữa, nếu dữ liệu hợp lệ sẽ tiến hành lưu xuống cơ sở dữ liệu. Ngược lại, dữ liệu không hợp lệ sẽ bắt người tạo phải nhập lại thông tin.

- Khi dữ liệu về mô tả chiến dịch được lưu ở cơ sở dữ liệu tập trung thành công thì DApp sẽ tiến hành tạo giao dịch có kèm mã băm đã tính toán trước đó cùng các thông tin như: mục tiêu gây quỹ, thời gian gây quỹ, hồ sơ giải ngân, RefID. Sau đó transaction được gửi tới Wallet Provider.
- Wallet Provider tiến hành kí cho transaction bằng khóa bí mật của người tạo. Sau đó transaction được gửi tới một nút trong mạng blockchain, để hợp đồng thông minh xử lý.
- Khi giao dịch được gửi tới, hợp đồng thông minh sẽ kiểm tra lại lần nữa các đầu vào, nếu dữ liệu hợp lệ thì dữ liệu được lưu. Ngược lại trả về thông báo lỗi cho người dùng.



Hình 3.10: Sơ đồ hoạt động tiến trình tạo lập chiến dịch

3.4.5 Chức năng giải ngân

3.4.5.1 Mục tiêu

Việc giải ngân được thực hiện khi chiến dịch gây quỹ đạt được mục tiêu gây quỹ. Và giải ngân đối với chiến dịch sẽ được phân làm hai loại:

- **Giải ngân một giai đoạn** - sau khi chiến dịch gây quỹ hoàn thành mục tiêu gây quỹ trong thời gian đặt ra, thì người tạo chiến dịch có thể gọi lệnh giải ngân và rút được tiền từ chiến dịch.
- **Giải ngân theo nhiều giai đoạn** - với một số chiến dịch có thể thực hiện theo nhiều giai đoạn khác nhau, thì việc giải ngân theo nhiều giai đoạn nhằm mục tiêu:
 - Buộc người tạo chiến dịch có trách nhiệm báo cáo tiến độ thực hiện chiến dịch.
 - Tăng cường quyền của người đóng góp bằng cách bỏ phiếu đồng ý xác nhận giải ngân cho chiến dịch theo từng giai đoạn.

3.4.5.2 Cách thức hoạt động

Chức năng giải ngân bao gồm các tiến trình:

- Tạo hồ sơ giải ngân.
- Bỏ phiếu đồng ý giải ngân.
- Gọi lệnh giải ngân.

Tiến trình tạo hồ sơ giải ngân: hồ sơ giải ngân sẽ được người tạo chiến dịch nhập vào lúc đăng ký chiến dịch gây quỹ, các thông tin trong hồ sơ giải ngân bao gồm:

- Số giai đoạn thực hiện giải ngân. Nếu giai đoạn giải ngân là 1 thì các thông tin bên dưới có thể bỏ trống.
- Số tiền cần cho mỗi giai đoạn (tổng tiền ở các giai đoạn sẽ bằng mục tiêu gây quỹ).
- Tùy chọn chế độ giải ngân, có 4 chế độ:
 - **Flexible** – đủ điều kiện giải ngân ở giai đoạn nào thì được rút tiền ở giai đoạn đó.
 - **Fixed** – muốn rút tiền ở giai đoạn tiếp theo thì giai đoạn trước đó phải đủ điều kiện giải ngân.
 - **TimingFlexible** – người tạo chiến dịch sẽ ấn định thời gian thực hiện cho mỗi giai đoạn, khi hết thời gian ấn định của giai đoạn hiện tại thì mới có thể thực hiện lệnh bỏ phiếu giải ngân và rút tiền cho giai đoạn tiếp theo.
 - **TimingFixed** – người tạo chiến dịch sẽ ấn định thời gian thực hiện cho mỗi giai

đoạn, khi hết thời gian ấn định của giai đoạn hiện tại thì mới có thể thực hiện lệnh bỏ phiếu giải ngân và rút tiền cho giai đoạn tiếp theo. Nếu giai đoạn hiện tại không đủ điều kiện giải ngân thì giai đoạn sau sẽ không được rút tiền.

Tiền trình bỏ phiếu giải ngân cho mỗi giai đoạn – tiền trình bỏ phiếu được đặc tả như sau:

- Chỉ có người đã đóng góp tiền cho chiến dịch thì mới có quyền bỏ phiếu cho chiến dịch đó.
- Việc bỏ phiếu đồng ý giải ngân được thực hiện cho từng giai đoạn giải ngân của chiến dịch, không bỏ phiếu cho toàn bộ chiến dịch.
- Với mỗi lá phiếu sẽ có hai tùy chọn: Đồng ý hoặc Không đồng ý giải ngân.
- Số phiếu “Đồng ý” đạt từ 50% trên tổng số người đã đóng góp vào chiến dịch và tổng số tiền mà những người đồng ý giải ngân phải đạt tỉ lệ trên 50% tổng số tiền mục tiêu gây quỹ thì được xem là đủ điều kiện giải ngân.

Tiền trình gọi lệnh giải ngân – việc gọi lệnh giải ngân sẽ có hai trường hợp:

- Đối với chiến dịch chỉ có một giai đoạn giải ngân: được rút toàn bộ số tiền gây quỹ được nếu hoàn thành mục tiêu gây quỹ trong thời gian đặt ra.
- Chiến dịch từ 2 giai đoạn thực hiện trở lên: giai đoạn đầu tiên sẽ được rút tiền mà không cần người đóng góp bỏ phiếu. Từ giai đoạn thứ hai trở đi, cần đạt điều kiện về bỏ phiếu giải ngân thì mới được rút tiền.

3.4.6 Hoàn tiền chiến dịch gây quỹ

3.4.6.1 Mục tiêu

Chức năng này giúp người đóng góp tiền vào chiến dịch có thể thực hiện hoàn tiền trong hai trường hợp:

- Trường hợp 1: hoàn tiền khi đang trong thời gian gây quỹ. Việc hoàn tiền này được thực hiện theo yêu cầu của người đóng góp.
- Trường hợp 2: hoàn tiền sau khi hết thời gian gây quỹ. Điều kiện để hoàn tiền trong trường hợp này là chiến dịch không đạt được mục tiêu gây quỹ trong thời gian đặt ra. Việc hoàn tiền này phải diễn ra một cách tự động. Tức người đóng góp không cần thực hiện bất kì thao tác gì.

3.4.6.2 Cách hoạt động

Phần này, nhóm tác giả tập trung vào việc hoàn tiền tự động khi chiến dịch kêu gọi quỹ thất bại (không đạt được mục tiêu kêu gọi quỹ).

Có hai giải pháp thiết kế thuật toán đáp ứng yêu cầu đặt ra:

- **Giải pháp 1 – chủ động:** thực hiện chạy chủ động lặp đi lặp lại một hàm nào đó trong hợp đồng thông minh để kiểm tra chiến dịch đã kết thúc và cập nhật các giá trị sau khi hết thời gian gây quỹ.
- **Giải pháp 2 – bị động:** không thực hiện bất kì thay đổi nào về giá trị token của người dùng được lưu trữ trong hợp đồng thông minh khi chiến dịch thất bại. Khi người dùng kiểm tra số dư hoặc thực hiện rút tiền sẽ tiến hành kiểm tra số dư thực tế dựa vào trạng thái thành công hay thất bại của các chiến dịch gây quỹ.

Với hai giải pháp được nêu trên, nhóm tác giả chọn giải pháp thứ 2 để thiết kế, với lí do sau:

- Chi phí của giải pháp thứ hai thấp hơn đáng kể so với giải pháp thứ nhất. Do việc kiểm tra số dư chỉ đơn thuần là đọc dữ liệu.
- Đặc điểm của blockchain là để thay đổi các giá trị đã lưu trữ thì cần tạo ra các transaction gọi đến các hàm trong hợp đồng thông minh thì hàm đó mới được khởi chạy và thay đổi dữ liệu. Do đó hiện tại, Ethereum chưa hỗ trợ việc lập lịch cho một giao dịch tự động chạy theo thời gian định trước. Vì vậy để hiện thực giải pháp 1 cần chạy một máy chủ thực hiện việc gửi transaction theo lịch định sẵn. Việc tạo một máy chủ ở đây có thể là một điểm chênh. Và với mỗi giao dịch được chạy tự động như vậy về lâu dài sẽ tốn một chi phí không nhỏ cho người vận hành.

Với giải pháp thứ hai được chọn, nhóm tác giả đề xuất công thức tính toán cho số dư token của người dùng trong hệ thống như sau:

$$T = T_0 - T_{donated}$$

Trong đó:

- T là số dư thực tế của người dùng.
- T_0 là số dư ban đầu người dùng gửi vào hợp đồng thông minh.
- $T_{donated}$ là số token mà người dùng đã đóng góp vào chiến dịch. Các chiến dịch ở đây bao gồm các chiến dịch đang kêu gọi gây quỹ và chiến dịch đã kêu gọi thành công. Không bao gồm chiến dịch kêu gọi thất bại.

3.5 Tổ chức dữ liệu

3.5.1 Dữ liệu phi tập trung

Cấu trúc hợp đồng thông minh được thể hiện ở hình 3.11. Cụ thể trong hệ thống có các hợp đồng thông minh sau:

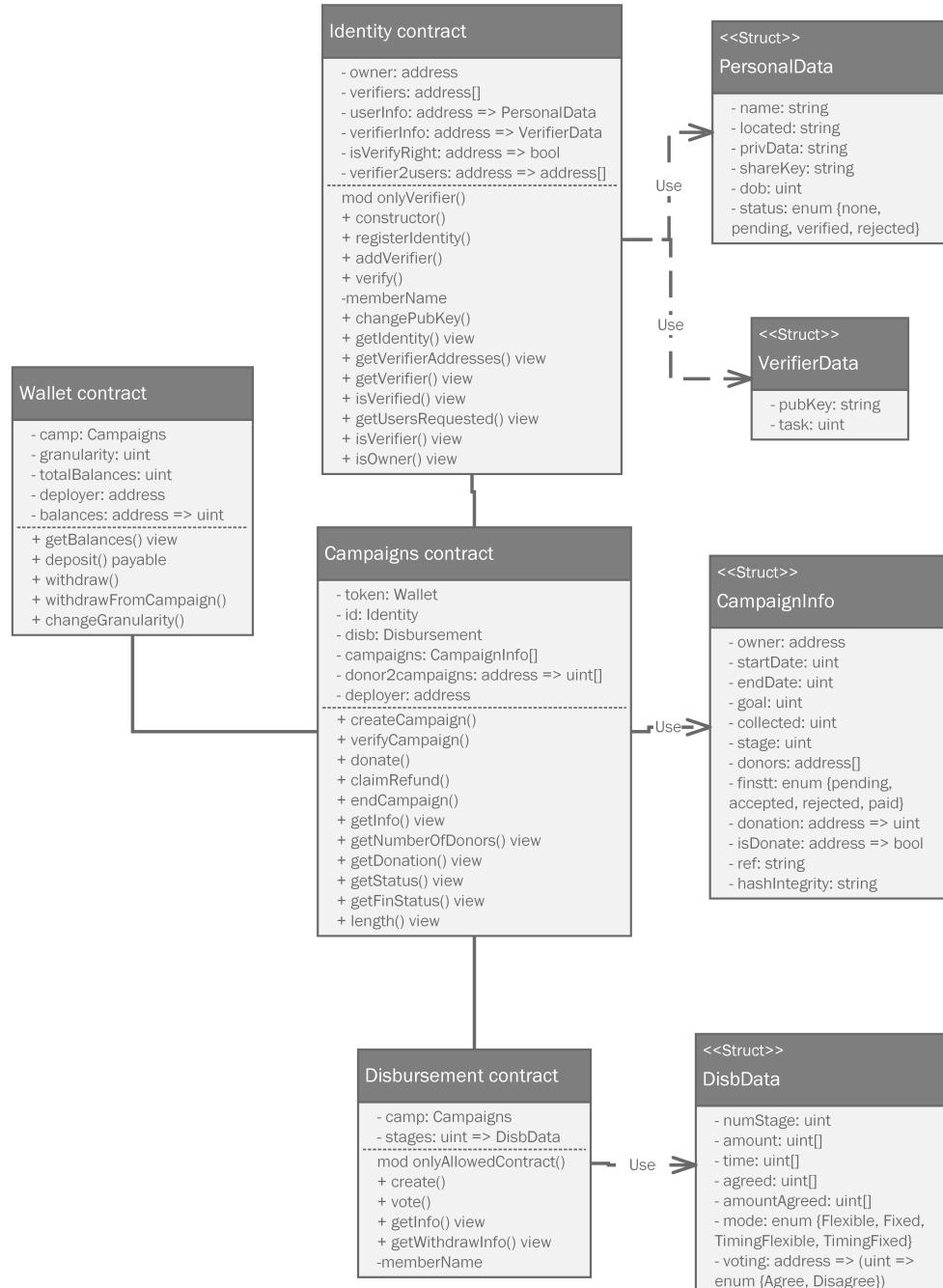
- **Wallet** – đây là hợp đồng chứa mã lưu trữ và xử lý các tác vụ liên quan đến tài chính trong hệ thống, bao gồm các tác vụ như nộp tiền và rút tiền trong hệ thống.
- **Campaigns** – là một contract chứa nhiều mã xử lý nhất trong hệ thống, bao gồm việc lưu trữ thông tin liên quan đến chiến dịch, các tác vụ như tạo chiến dịch, đóng góp tiền vào một chiến dịch, giải ngân.
- **Identity** – chứa mã lưu trữ và xử lý các tác vụ liên quan đến thông tin định danh. Các tác vụ trên contract này như: đăng ký hồ sơ định danh, duyệt hồ sơ định danh.
- **Disbursement** – chứa mã lưu trữ thông tin giải ngân của chiến dịch, tác vụ bù phiếu giải ngân được contract này xử lý.

3.5.2 Dữ liệu tập trung

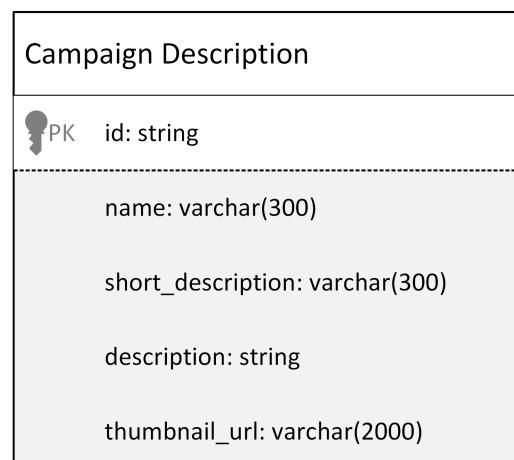
Các dữ liệu được lưu tại cơ sở dữ liệu tập trung bao gồm các thông tin mô tả chiến dịch:

- Tên chiến dịch.
- Thông tin ngắn gọn về chiến dịch.
- Mô tả chi tiết về chiến dịch.
- Hình ảnh mô tả về chiến dịch dưới dạng đường dẫn.

Bảng thiết kế cơ sở dữ liệu được mô tả ở hình 3.12.



Hình 3.11: Kiến trúc hợp đồng thông minh



Hình 3.12: Thiết kế cơ sở dữ liệu tập trung

Chương 4

HIỆN THỰC VÀ ĐÁNH GIÁ HỆ THỐNG

4.1 Hiện thực

4.1.1 Môi trường hiện thực

Để thuận tiện trong việc đóng gói và triển khai ứng dụng, nhóm tác giả sử dụng công nghệ Container để hỗ trợ, đại diện tiêu biểu cho công nghệ Container mà nhóm tác giả chọn để sử dụng là Docker¹.

Công nghệ Docker container giúp người phát triển có thể triển khai ứng dụng ở bất kì đâu miễn là thiết bị đó có thể chạy được docker. Các ứng dụng được đóng gói trong container, có thể kiểm tra, xóa bất kì container nào. Các ứng dụng được triển khai dễ dàng và đồng nhất giữa các môi trường khác nhau.

Phiên bản docker và ứng dụng kèm theo mà nhóm tác giả triển khai bao gồm:

- **Docker Engine:** 19.03.3
- **Docker Compose:** 1.21.0

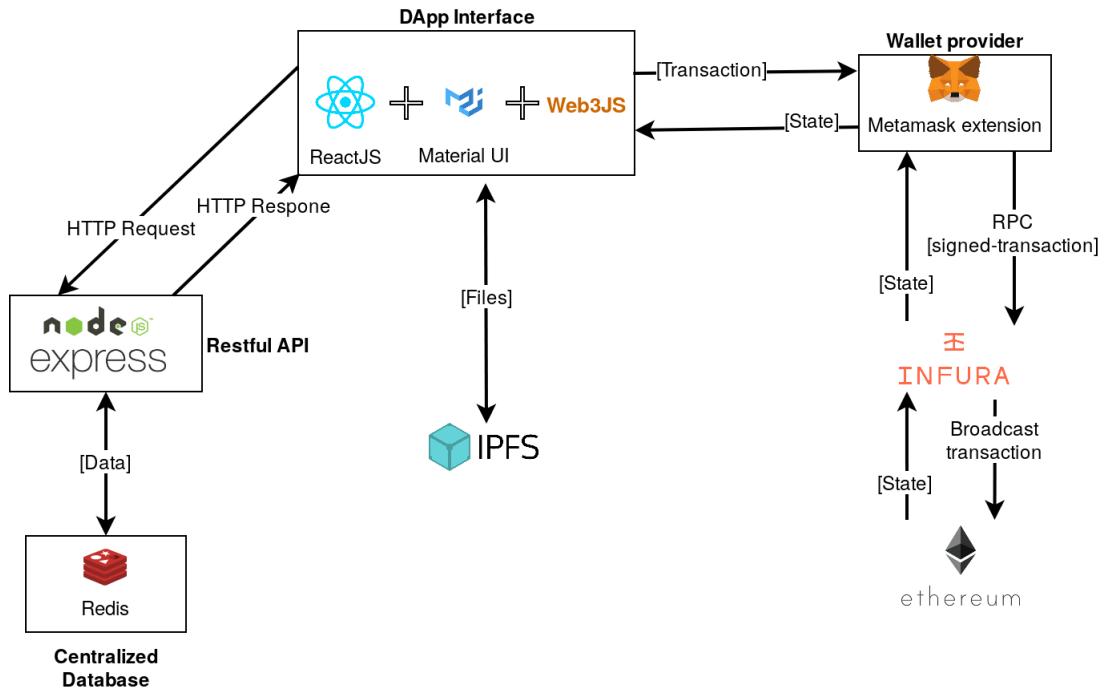
4.1.2 Các công nghệ được sử dụng

Dựa vào kiến trúc hệ thống ở hình 3.2, nhóm tác giả tiến hành chọn các công nghệ để hiện thực. Sơ đồ tổng quan về các công nghệ mà nhóm tác giả hiện thực được thể hiện ở hình 4.1.

Các công nghệ được sử dụng bao gồm:

- Phần giao diện người dùng (front-end): nhóm tác giả sử dụng **ReactJS/NodeJS** kết

¹<https://www.docker.com>



Hình 4.1: Sơ đồ hiện thực hệ thống

hợp **Material UI** để tạo giao diện ứng dụng web cho người dùng. Để tương tác với các hợp đồng thông minh, nhóm tác giả sử dụng thư viện Web3js và trình mở rộng Ví Metamask.

- Phần back-end được chia làm hai phần như sau:
 - Kiến trúc phi tập trung (decentralized): nhóm sử dụng ngôn ngữ solidity để xây dựng các smart contract kết hợp IPFS để thực hiện lưu trữ các dữ liệu phi tập trung.
 - Kiến trúc tập trung (centralized): công nghệ NodeJS kết hợp với Redis để tổ chức và tương tác với dữ liệu tập trung.

Với hợp đồng thông minh và ngôn ngữ Solidity, nhóm tác giả sử dụng bộ công cụ Truffle framework để xây dựng và triển khai các mã hợp đồng thông minh.

4.1.2.1 Công nghệ ReactJS/NodeJS

ReactJS là một thư viện để hỗ trợ cho việc phát triển giao diện của người dùng. Đây là một trong những thư viện front-end nổi tiếng với hơn 141.000 sao đánh giá với hơn 2.8 triệu người dùng trên dịch vụ lưu trữ mã nguồn Github² (số liệu cập nhật tháng 12 năm 2019). Thư viện

²<https://github.com>

này được hỗ trợ và phát triển bởi Facebook³ và Instagram⁴ cùng với cộng đồng các nhà phát triển trên toàn thế giới và đang được phát triển từng ngày. ReactJS cung cấp tốt hơn về trải nghiệm người dùng và đồng thời có nhiều thư viện bên thứ ba hữu ích khác được phát triển kèm theo.

ReactJS được thiết kế và phát triển theo kiến trúc các component⁵. Theo như tài liệu từ Facebook, định nghĩa React là một thư viện dành cho phát triển các mô đun cho giao diện người dùng. Về cơ bản, React cho phép các nhà lập trình phát triển các ứng dụng web lớn, phức tạp và đồng thời có thể thay đổi dữ liệu mà không cần tải lại trang. React sử dụng DOM ảo, điều này có tác dụng tăng hiệu suất kết xuất trang web, đồng thời nâng cao trải nghiệm của người dùng, cũng như rất dễ dàng lập trình phát triển sản phẩm.

Node.js hay còn được gọi là **Node** – là một nền tảng chạy trên môi trường Javascript hay còn được gọi là một nền tảng chạy trên môi trường V8 JavaScript runtime. Công nghệ V8 đã được triển khai hầu hết ở C và C++, công nghệ này tập trung chủ yếu vào hiệu suất và ít tiêu tốn bộ nhớ. Hầu hết V8 hỗ trợ chủ yếu Javascript trên trình duyệt (chú ý nhất là Google Chrome), mục đích hỗ trợ của Node đó chính là hỗ trợ các tiến trình có thời gian chạy dài. Không giống như những ngôn ngữ khác, Node không hỗ trợ đa luồng, nhưng hỗ trợ xử lý bất đồng bộ **nodejs-performance**. Nhờ vào tính năng xử lý bất đồng bộ này, Node còn được biết đến là một nền tảng xử lý nhanh chóng hàng ngàn yêu cầu đồng thời, chịu tải cao, tốc độ thực thi và khả năng mở rộng tốt.

Đánh giá về hiệu suất của các ứng dụng chạy Node.js, nhóm tác giả có tham khảo nghiên cứu của tác giả Robert Ryan McCune về đo lường hiệu suất của ứng dụng Node.js **mccune2011node**. Trong bài nghiên cứu, tác giả Robert Ryan McCune đã thực hiện phần đánh giá trên một máy ảo **Ubuntu 11.10** được tạo bởi VMWare Fusion 4.0.2 trên iMac OS X 10.6.8 với 8GB RAM và 3.06Ghz Intel Core 2 Duo, máy ảo này được cấu hình với bộ nhớ là **2GB RAM**, và một bộ vi xử lý lõi kép. Tác giả đã thực hiện đồng thời 100 và 1000 yêu cầu đến máy chủ chạy bằng Node và thu được kết quả là thời gian trả lời từ máy chủ chưa tới 70ms. Tương tự thực hiện 500 yêu cầu đồng thời và thực hiện tổng cộng 10.000 yêu cầu thì thời gian phản hồi là dưới 140ms.

4.1.2.2 Material UI framework

Material UI là một thư viện được *Google*⁶ viết dành riêng cho ReactJS, bao gồm tập hợp nhiều các thành phần được xây dựng và thiết kế theo phong cách Material. Với hơn 52.8

³<https://www.facebook.com>

⁴<https://www.instagram.com>

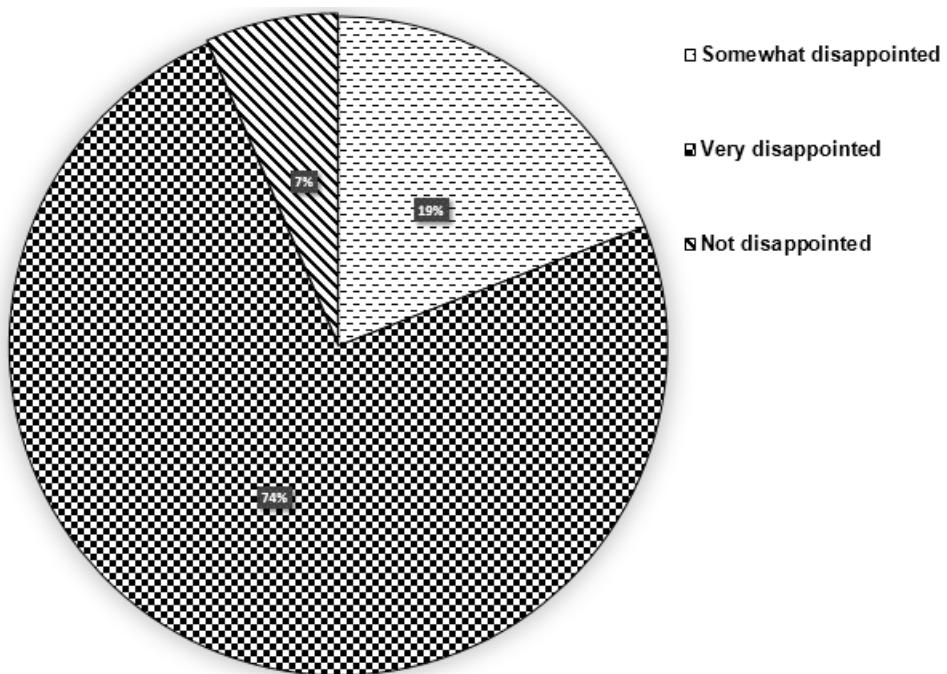
⁵là một kiểu kiến trúc trong phát triển phần mềm, chia ứng dụng ra thành các thành phần, bộ phận không phụ thuộc lẫn nhau và có thể tái sử dụng các thành phần này khi cần thiết

⁶<https://www.google.com>

nghìn sao đánh giá trên cộng đồng Github, và được sử dụng bởi hơn 139.000 dự án khác nhau, Material là một trong những thư viện UI được nhiều người sử dụng nhất trên thế giới. Giao diện của Material cũng tương đồng với các sản phẩm của Google như: *Gmail*, *Google tìm kiếm*, *Google Form*,...

Material UI hiện tại cũng đang được sử dụng bởi *NASA*, *UNIQLO*, *shutterstock*,...

Trong cuộc khảo sát vào năm 2019 của Oliver được đăng trên trang Medium⁷ đã chỉ ra rằng, có đến 74.4% của 734 người khảo sát cho rằng sẽ thất vọng khi không còn sử dụng thư viện này. Biểu đồ thể hiện thái độ người khảo sát khi không còn được sử dụng thư viện Material ở hình 4.2.



Hình 4.2: Biểu đồ thể hiện thái độ người dùng khi không được sử dụng thư viện Material

Cũng tại bài viết, tác giả đã ghi nhận nhận những lợi ích từ người khảo sát cho rằng yêu tố tập trung vào luồng xử lý, tiết kiệm thời gian, dễ dùng, ... và có đến 70% dùng thư viện này để xây dựng dashboard, 40% để thiết kế hệ thống, 35% dùng để thiết kế các trang doanh nghiệp.

4.1.2.3 Cơ sở dữ liệu Redis

Redis là một cơ sở dữ liệu thường được xếp vào nhóm cơ sở dữ liệu NoSQL, được phát triển vào năm 2009. Khác với những cơ sở dữ liệu khác, Redis lưu trữ dữ liệu trên RAM của máy chủ, điều này làm cho việc truy xuất giá trị nhanh hơn so với cách lưu trữ truyền thống – lưu trữ trên ổ cứng. Sau một thời gian, các bản ghi được lưu trên RAM này sẽ được lưu xuống ổ

⁷Nguồn: <https://medium.com/material-ui/2019-material-ui-developer-survey-results-c9589434bbcf>

cứng nhằm tiết kiệm tài nguyên bộ nhớ RAM.

Các đặc điểm của Redis **6106531**:

- (i) Redis là một dạng lưu trữ dữ liệu *key-value*, như được nói ở trên khi Redis chạy, dữ liệu sẽ được lưu trữ trong bộ nhớ, do đó nó có thể xử lý hơn 100.000 thao tác đọc và ghi mỗi giây.
- (ii) Redis hỗ trợ nhiều dạng lưu trữ như *List* và *Set*,...
- (iii) Giá trị lớn nhất lưu trên Redis là 1GB.
- (iv) Nhược điểm của Redis là dung lượng của cơ sở dữ liệu bị giới hạn bởi bộ nhớ vật lý, do đó Redis không nên dùng làm cơ sở dữ liệu cho các dự án lớn và khả năng mở rộng kém.

Với các đặc điểm trên, Redis phù hợp cho việc cung cấp hiệu suất cao cho lượng dữ liệu nhỏ.

4.1.2.4 IPFS

InterPlanetary File System (IPFS) là một giao thức chia sẻ tệp tin được phân tán ngang hàng Peer-to-Peer (P2P). Giao thức này cho phép người dùng chia sẻ các tệp tin ngang hàng với nhau mà không cần có sự xuất hiện của máy chủ. Các dữ liệu khi người dùng tải lên sẽ được băm ra và đồng thời sinh ra mã băm của dữ liệu ấy. Với các dữ liệu giống nhau thì sẽ tạo ra những hàm băm giống nhau, do đó IPFS sẽ hạn chế được sự trùng lặp.

IPFS có thể giải quyết vấn đề về lưu trữ dữ liệu lớn cho các ứng dụng blockchain bằng cách sử dụng blockchain để lưu trữ địa chỉ của dữ liệu được ra bằng IPFS (chính là mã băm nhận diện cho tập tin) và đặt địa chỉ bất biến này cho một giao dịch trong blockchain. Không những giải quyết vấn đề lưu trữ, IPFS còn giải quyết vấn đề về băng thông bằng cách phân tán nội dung đến hệ thống P2P. Do đó, khi truy cập tập tin nào đó bằng cách dùng hàm băm, nút gần nhất với tệp sẽ phản hồi và gửi tệp cho người yêu cầu. Như đã được chứng minh từ trước **5235364**, một hệ thống phân tán P2P có thể tiết kiệm lên đến 60% so với hệ thống truyền thống. Đồng thời, IPFS còn tăng tính bảo mật và chống lại các cuộc tấn công từ chối dịch vụ và giới hạn lại kiểm duyệt vì không có địa chỉ IP cụ thể của máy chủ bị chặn **8441990**.

4.1.2.5 Bộ công cụ Truffle framework

Truffle là một bộ công cụ để phát triển các ứng dụng phi tập trung trên mạng Ethereum. Khi sử dụng truffle, theo tài liệu kỹ thuật⁸ của Truffle mô tả framework này có các tính năng sau:

- Tích hợp các tính năng biên dịch, liên kết, triển khai và quản lý mã nhị phân các hợp đồng thông minh.

⁸<https://www.trufflesuite.com/docs>

- Tự động kiểm thử các contract để quá trình phát triển nhanh hơn.
- Khung triển khai các hợp đồng thông minh có thể tùy biến và mở rộng.
- Quản lí mạng để triển khai hợp đồng thông minh đến bất kì mạng công khai hay riêng tư nào.
- Quản lí gói với EthPM và NPM, sử dụng tiêu chuẩn ERC190⁹.
- Tương tác trực tiếp với hợp đồng thông minh thông qua giao diện console.

4.1.2.6 Thư viện web3js

Web3js là một thư viện được viết bằng *Javascript*, thư viện cung cấp các API cần thiết cho lập trình viên tương tác với mạng blockchain Ethereum cũng như là hợp đồng thông minh trên Ethereum thông qua giao tiếp Remote Procedure Call (RPC)¹⁰.

Theo như tài liệu của thư viện Web3js¹¹, các hàm trong *web3-eth* dùng để tương tác với mạng blockchain và hợp đồng thông minh như là lấy thông tin địa chỉ của tài khoản, chọn lứa mạng,... *web3-ssh* cung cấp giao thức để giao tiếp P2P và broadcast. *web3-bzz* được dành cho giao thức swarm¹², lưu trữ tập trung. Cuối cùng là *web-utils* chứa đựng các hàm tiện ích cho dự án DApp.

4.1.2.7 Ví Metamask

Metamask¹³ là một ví Ethereum, được phát triển dưới dạng trình mở rộng trên trình duyệt. Metamask là ứng dụng cầu nối cho phép người dùng truy cập vào các ứng dụng web phi tập trung và tương tác với mạng blockchain đơn giản hơn mà không cần phải chạy một nút Ethereum đầy đủ.

Metamask cung cấp giao diện người dùng để quản lý các khóa bí mật, các giao dịch, số dư Ethereum, kí transaction, Metamask có giao diện dễ dùng, hỗ trợ hầu hết các trình duyệt hiện nay.

4.1.3 Các bước hiện thực

Phần hiện thực hệ thống được viết thành kịch bản và thực hiện tự động bằng trình Docker Compose, hệ thống sẽ được chia thành nhiều dịch vụ, bao gồm:

- **smartcontract** – thực hiện chức năng biên dịch và triển khai các hợp đồng thông minh lên mạng blockchain, sau đó trả về các tệp json chứa Application Binary Interface

⁹<https://github.com/ethereum/EIPs/issues/190>

¹⁰Remote Procedure Call (RPC) tạm dịch là các cuộc gọi thủ tục từ xa, đây là một phương pháp dùng để trao đổi dữ liệu theo kiến trúc yêu cầu - phản hồi.

¹¹<https://web3js.readthedocs.io>

¹²swarm - một mô hình kiến trúc phân tán.

¹³<https://metamask.io>

(ABI)¹⁴ và địa chỉ của hợp đồng thông minh. Các tệp json này được front-end sử dụng để kết nối với hợp đồng thông minh.

- **client** – kết xuất và triển khai giao diện người dùng.
- **store_centralized_data** – xây dựng và triển khai Restful API để xử lý đọc và ghi dữ liệu ở cơ sở dữ liệu tập trung.
- **redis** – cơ sở dữ liệu Redis, lưu trữ các thông tin mô tả về chiến dịch.

Nội dung tệp **docker-compose.yml**¹⁵ triển khai các dịch vụ trong hệ thống như sau:

```
version: '3.4'
services:
  smartcontracts:
    build: ./smartcontracts/
    volumes:
      - contracts:/app/build/
  client:
    build: ./client/
    ports:
      - 3000:3000
    volumes:
      - ./client/src/:/app/src/
      - contracts:/app/src/contracts/:ro
    depends_on:
      - smartcontracts
  store_centralized_data:
    build: ./store_centralized_data/
    ports:
      - 8080:8080
    volumes:
      - ./store_centralized_data/src/:/app/src/
    depends_on:
      - redis
  redis:
    image: redis:alpine
    command: redis-server --requirepass 12345678 --appendonly yes
    volumes:
      - data:/data
  volumes:
    data:
    contracts:
```

Các bước để hiện thực hệ thống như sau:

- **Bước 1:** cấu hình thông số các service.
- **Bước 2:** tạo các service.

¹⁴là cách tiêu chuẩn để tương tác với các hợp đồng trong hệ sinh thái Ethereum, cả từ bên ngoài blockchain và cho tương tác giữa các hợp đồng với nhau.

¹⁵docker-compose.yml là tệp mặc định được sử dụng bởi Docker Compose

- **Bước 3:** chạy các service.

Chi tiết các bước trên được mô tả ở mục 4.1.3.1, 4.1.3.2, 4.1.3.3.

4.1.3.1 Cấu hình thông số các service

Để các service chạy chính xác, nhóm tác giả tạo ra các biến môi trường được lưu ở file **.env** trong thư mục của từng service trong bộ mã nguồn chương trình kèm theo khóa luận này để phục vụ cho việc cấu hình.

Cấu hình service smartcontracts, chỉnh sửa các biến sau trong tệp **smartcontracts/.env**:

- MNEMONIC – biến chứa giá trị mnemonic của tài khoản nhân viên vận hành hệ thống, mnemonic là một chuỗi 12 kí tự gợi nhớ, chuỗi này gắn liền với các tài khoản trên ethereum. Tài khoản mặc định gắn với mnemonic nhập vào sẽ là tài khoản triển khai các hợp đồng thông minh lên mạng blockchain.
- INFURA_API_KEY – khóa để sử dụng API của Infura¹⁶ để tương tác với một nút trong mạng blockchain để triển khai các hợp đồng thông minh lên mạng. Để có khóa này, có thể đăng ký sử dụng API của Infura. Giá trị mặc định:

417dc4db619c43798109b08709985882

Cấu hình service client, chỉnh sửa các biến trong tệp **client/.env**:

- REACT_APP_STORE_CENTRALIZED_API – là địa chỉ máy chủ Restful API trong mô hình hệ thống, định dạng “*http://[IP|DOMAIN]:PORT/*”
- REACT_APP_DEFAULT_NETWORK – địa chỉ để kết nối đến một nút mạng blockchain mặc định để lấy dữ liệu, phục vụ việc hiển thị thông tin chiến dịch khi không có ví Metamask. Giá trị mặc định:

<https://ropsten.infura.io/v3/b56a34eee35e491691629e3412875afa>

- REACT_APP_RECAPTCHA_ENABLE – biến cờ để xác định việc sử dụng captcha trong trang đăng ký chiến dịch. Với giá trị “1” là xác định sử dụng dịch vụ captcha, “0” là tắt captcha. Giá trị mặc định: “0”.
- REACT_APP_RECAPTCHA_SITEKEY – phần captcha trên trang đăng ký chiến dịch được nhóm tác giả sử dụng là reCAPTCHA¹⁷ của Google, nên để sử dụng được thành phần này trong hệ thống, cần đăng ký dịch vụ ReCaptcha của Google. Sau đó thực hiện lấy khóa captcha tương ứng để sử dụng. Nếu captcha không được sử dụng thì có thể bỏ qua biến này.

¹⁶<https://infura.io>

¹⁷<https://www.google.com/recaptcha/intro/v3.html>

Cuối cùng là cấu hình cho service **store_centralized_data**:

- PORT_LISTEN – cổng mà máy chủ sẽ lắng nghe. Giá trị mặc định: “**8080**”.
- RECAPTCHA_ENABLE – biến cờ để xác định việc sử dụng captcha trong trang đăng ký chiến dịch. Với giá trị “**1**” là xác định sử dụng dịch vụ captcha, “**0**” là tắt captcha. Giá trị mặc định: “**0**”.
- RECAPTCHA_SECRET_KEY – phần khóa captcha này liên kết với captcha ở phía service **client**. Tuy nhiên ở service này sẽ nhận kết quả captcha từ client gửi và xử lý. Nên khóa ở đây là khóa bí mật của dịch vụ *reCAPTCHA*.
- Các biến REDIS_HOST, REDIS_PORT, REDIS_PASSWORD là thông tin để kết nối đến dịch vụ redis. Giá trị mặc định được gán cho các biến này gồm: “redis”, “6379”, “12345678”.

4.1.3.2 Tạo các service

Để các service có thể chạy được thì cần cài đặt môi trường và thư viện tương ứng. Để đơn giản quá trình cài đặt môi trường, thư viện phức tạp, nhóm tác giả đã tiến hành viết sẵn kịch bản để cài đặt môi trường cần thiết. Kịch bản này được tạo ra bằng tệp **Dockerfile** và lưu vào từng thư mục của từng service.

Tệp **Dockerfile** của services **smartcontracts** có nội dung như sau:

```
# BUILDER IMAGE
FROM node:8-alpine AS builder

# Create app directory and set to working directory
WORKDIR /app

# Install dev dependencies
RUN apk add --no-cache git python make g++

# Install dependencies
COPY yarn.lock package.json /app/
RUN yarn

# RUN TIME IMAGES
FROM node:8-alpine

# Create app directory and set to working directory
WORKDIR /app

# Copy from builder image
COPY --from=builder /app .

# Bundle source code
COPY . /app/

# Build and deploy contract to Ethereum network
```

```
RUN yarn truffle migrate --network ropsten
```

Tiếp theo là tệp **Dockerfile** của services **client** có nội dung như sau:

```
# BUILDER IMAGE
FROM node:8-alpine AS builder

# Create app directory and set to working directory
WORKDIR /app

# Install dev dependencies
RUN apk add --no-cache python make g++

# Install dependencies
COPY yarn.lock package.json /app/
RUN yarn

# RUN TIME IMAGES
FROM node:8-alpine

EXPOSE 3000

# Create app directory and set to working directory
WORKDIR /app

# Copy from builder image
COPY --from=builder /app .

# Bundle source code
COPY . /app/

CMD yarn start
```

Cuối cùng là tệp **Dockerfile** của services **stored_centralized_data**:

```
FROM node:8-alpine

EXPOSE 8080
ENV DIR /app

# Create app directory and set to working directory
WORKDIR ${DIR}

# Install dependencies
COPY yarn.lock package.json ${DIR}/
RUN yarn

# Bundle source code
COPY . ${DIR}/

CMD yarn start
```

4.1.3.3 Chạy các service

Chạy lệnh sau ở thư mục gốc của ứng dụng (thư mục chứa tệp docker-compose.yml) để tiến hành xây dựng và khởi chạy các service:

```
$ docker-compose up
```

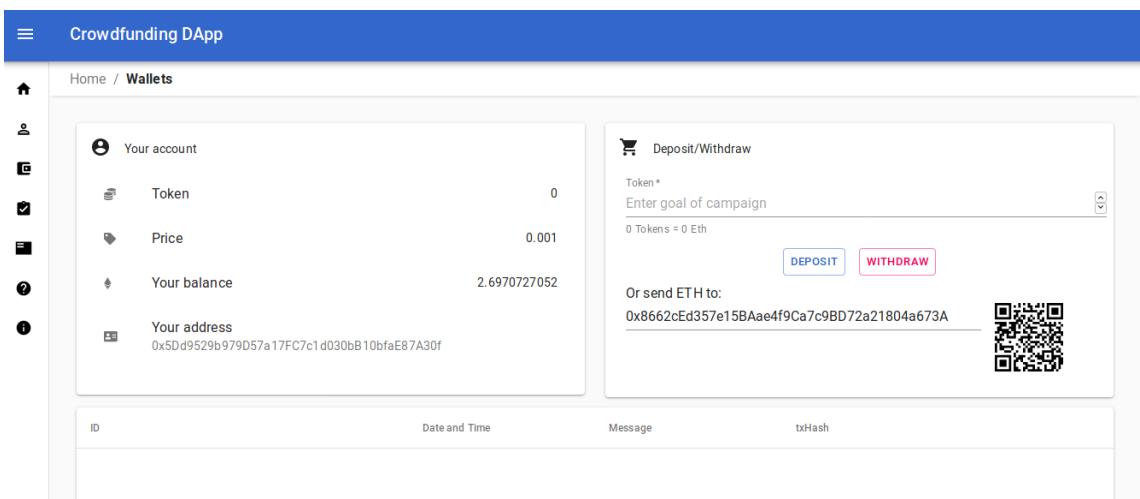
Sau khi chạy lệnh trên, các service có mở cổng để lắng kết nối bao gồm:

- **client** với cổng mặc định được định nghĩa là **3000**.
- **stored_centralized_data** với cổng mặc định được định nghĩa là **8080**.
- **redis** với cổng mặc định **6379** được mở trong phạm vi kết nối giữa các container với nhau.

4.1.4 Kết quả hiện thực

Kết quả sau khi hiện thực được trình bày theo quy trình từ đăng ký hồ sơ định danh đến khi một chiến dịch gây quỹ được tạo thành công.

Kết quả màn hình của trang nộp tiền / rút tiền trong hệ thống được thể hiện ở hình 4.3, người đóng góp sẽ thực hiện nộp tiền vào hệ thống để đóng góp cho các chiến dịch gây quỹ.

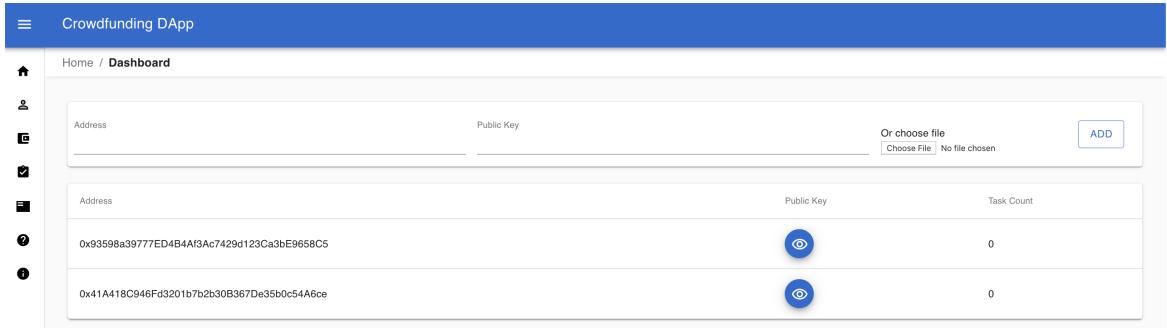


Hình 4.3: Màn hình giao diện trang nộp tiền và rút tiền

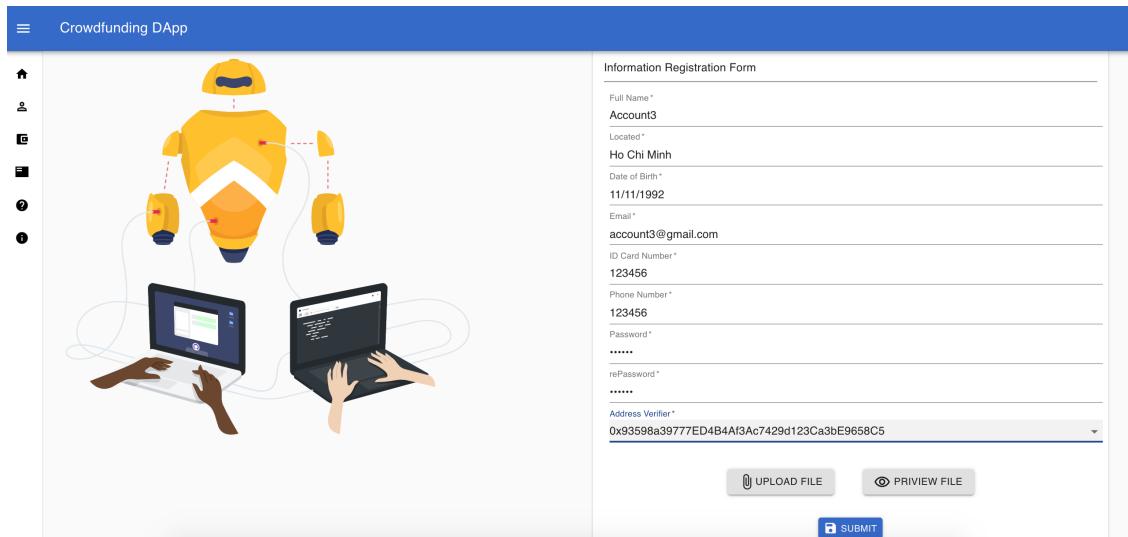
Nhân viên vận hành tiến hành thêm vào các nhân viên xác minh, ảnh chụp giao diện người dùng của trang thêm nhân viên xác minh ở hình 4.4.

Tiếp theo, người tạo chiến dịch tiến hành đăng ký hồ sơ định danh, giao diện trang đăng ký thông tin định danh dành cho người tạo chiến dịch được thể hiện ở hình 4.5. Người tạo chiến dịch sẽ nhập các thông tin cần thiết theo mẫu, sau đó nhấn vào nút “**Submit**”.

Sau đó, một hộp thoại sẽ xuất hiện để xác nhận lại các điều khoản, nếu chấp nhận các điều



Hình 4.4: Màn hình giao diện trang thêm nhân viên xác minh



Hình 4.5: Màn hình giao diện trang đăng ký định danh

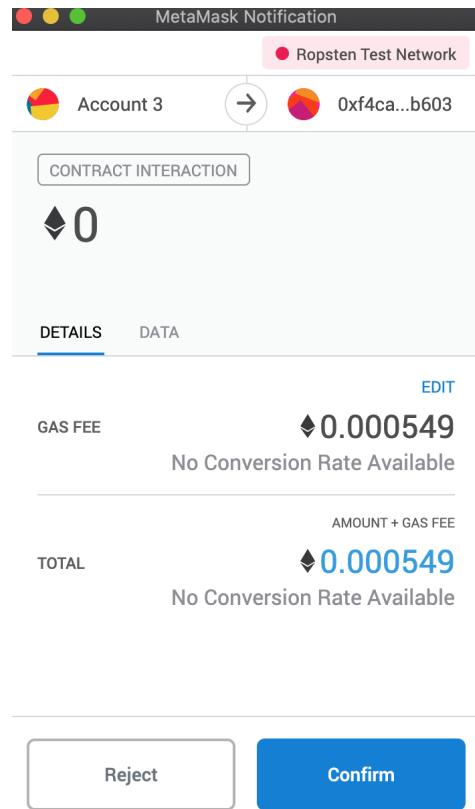
khoản thì sẽ tiến hành tạo transaction. Sau đó transaction được kí bởi Metamask, một hộp thoại khác được hiển thị để xác nhận phía Metamask như ở hình 4.6.

Transaction sau khi được kí sẽ được gửi đến mạng blockchain, có thể xem chi tiết transaction trên Etherscan¹⁸, hình 4.7 là ảnh chụp chi tiết transacion.

Tiếp theo, nhân viên xác minh tiến hành kiểm tra các danh sách các hồ sơ định danh, màn hình hiển thị danh sách người dùng đã đăng ký hồ sơ định danh được thể hiện ở hình 4.8.

Ứng với từng địa chỉ, ấn nút “VIEW” để xem chi tiết thông tin ở từng hồ sơ. Sau đó, một hộp thoại hiện lên chứa các thông tin như ở hình 4.9. Để xem thông tin bí mật chia sẻ từ người tạo lập hồ sơ, nhân viên xác minh cần nhập khóa bí mật của mình. Sau khi xem xét tất cả thông tin cần thiết, nhân viên xác minh có thể chấp nhận hồ sơ đó bằng cách nhấn vào nút “ALLOW” hoặc từ chối với nút “REJECT”. Hồ sơ định danh được chấp nhận thì người tạo chiến dịch mới được quyền tạo chiến dịch gây quỹ.

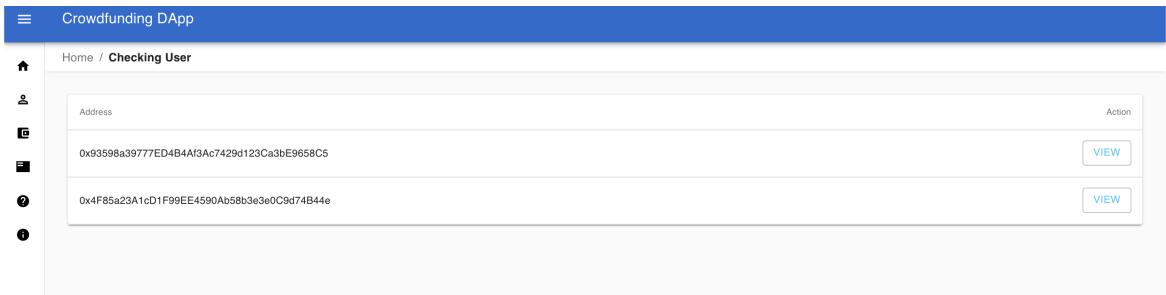
¹⁸<https://etherscan.io>



Hình 4.6: Màn hình hộp thoại xác nhận kí transaction trên Metamask

Overview	Internal Transactions	Event Logs (1)	State Changes
[This is a Ropsten Testnet transaction only]			
② Transaction Hash:	0xa64baa0200e773f35e61b219579b6e3348276956b78c18f09f60d941d651039b		
② Status:	Success		
② Block:	6960906	2 Block Confirmations	
② Timestamp:	1 min ago (Dec-13-2019 03:58:33 PM +UTC)		
② From:	0x4f85a23a1cd1f99ee4590ab58b3e3e0c9d74b44e		
② To:	Contract 0x13b2775c592e02c5dad412c75c6de873aea934f2		
② Value:	0 Ether (\$0.00)		
② Transaction Fee:	0.000385339 Ether (\$0.000000)		
Click to see More ↓			

Hình 4.7: Ảnh chụp chi tiết transaction trên etherscan



Hình 4.8: Giao diện trang hiển thị danh sách hồ sơ định danh

User's Information

This status's profile: pending

Full Name
👤 Account3

Located
📍 Ho Chi Minh

Date of birth
📅 Wed Nov 11 1992

To see user's private data, please input your private key

Private Key
OTP

Choose File No file chosen DECRYPT

ALLOW REJECT

Hình 4.9: Màn hình hiển thị thông tin định danh người dùng

Nếu hồ sơ định danh được xét duyệt, người tạo chiến dịch có thể đăng ký chiến dịch gây quỹ. Giao diện người dùng của trang đăng ký chiến dịch như ở hình 4.10.

Sau khi người tạo chiến dịch gửi transaction đi thành công thì chiến dịch đó tạm thời được gán trạng thái là đang chờ xét duyệt. Nhân viên xác minh sẽ tiến hành đọc thông tin chiến dịch và xét duyệt chiến dịch đó. Màn hình hiển thị thông tin chiến dịch chờ xét từ nhân viên xác minh được thể hiện ở hình 4.11, lưu ý trang này chỉ dành cho nhân viên xác minh.

Chỉ có những chiến dịch nào được xét duyệt cho đồng ý đóng góp mới được xuất hiện trong danh sách chiến dịch cho người đóng góp thấy. Hình 4.12 là ảnh chụp giao diện danh sách các chiến dịch đã được xác minh để người đóng góp có thể nhìn thấy được. Tại danh sách các chiến dịch, người đóng góp có thể thấy được các thông tin tổng quát của chiến dịch như thời gian bắt đầu, thời gian kết thúc chiến dịch, trạng thái của chiến dịch,

The screenshot shows the 'Create campaign' section of a crowdfunding application. On the left, there's a sidebar with icons for Home, Profile, Wallet, Help, and Info. The main area has a title 'Create campaign'. It includes fields for 'Name*' (with placeholder 'Enter name of campaign' and character limit 'Min: 30, Max: 300 characters'), 'Short description*' (placeholder 'Enter short description', note 'Min: 100, Max: 300 characters. Short description as slogan of campaign, it will be display on homepage.'), and a large 'Description' text area with placeholder 'Tell me a story' (note 'Min: 250, Max: 10000 characters. You can type as Markdown format'). There's a 'PREVIEW' button next to the description area. Below these are fields for 'Image thumbnail url*' (placeholder 'Enter url of thumbnail image', note 'Thumbnail image is best with size 286x180'), 'Goal*' (placeholder 'Enter goal of campaign', note 'Goal range: 1000-1.000.000.000 (Testing: min 1000 tokens)'), 'Deadline*' (placeholder 'Enter number of days', note 'This is time end campaign (days). Range: 15 - 180 days (In testing, min: 1 minutes)'), and 'Numstage' (input field '1', note 'This is number stage of project'). At the bottom is a reCAPTCHA box with 'I'm not a robot' and a 'CREATE CAMPAIGN' button.

Notes

Notes 1: A newly created campaign will need to wait for accept. The status of the campaign will be PENDING. During this time the campaign will not be able to perform any transactions.

In current, for testing, we set default for new campaign is Accepted.

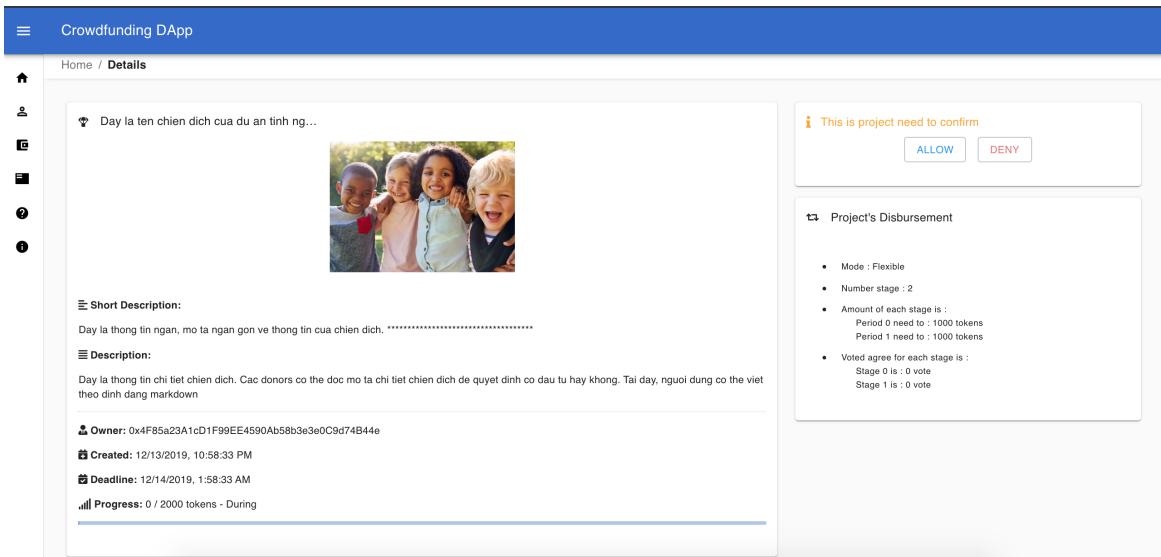
Notes 2: After the campaign was accepted, investors can donate for campaign. You (creator campaign) only can withdraw amount of campaign if campaign successful

Notes 3: A succeed campaign is reach goal and meet deadline.

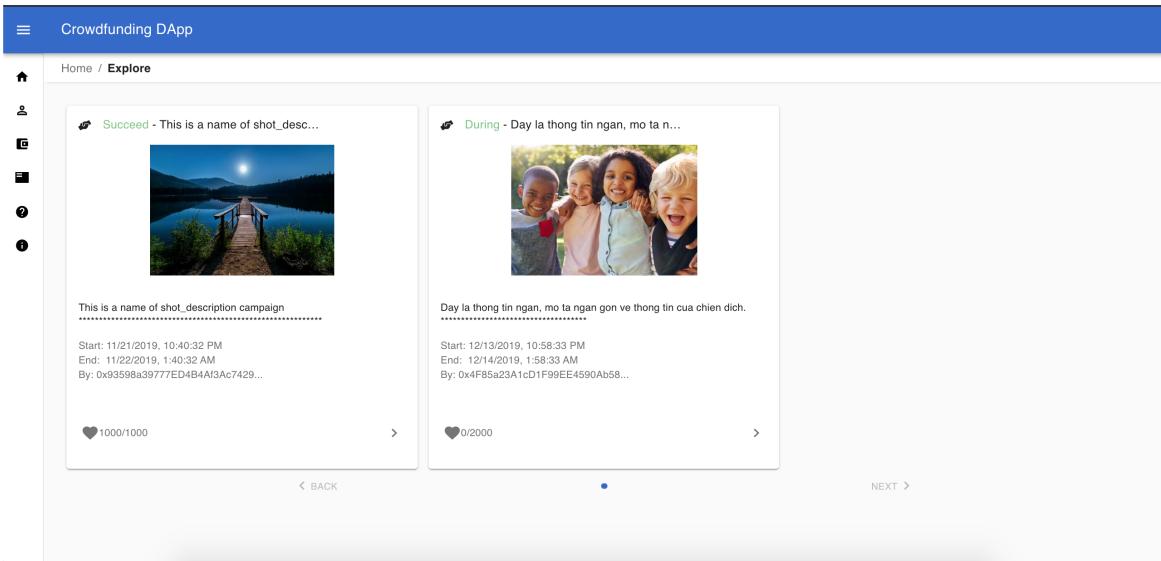
Notes 4: Any investors also can claim refund during campaign and when campaign failed. But NOT in succeed campaign

Hình 4.10: Giao diện trang đăng ký chiến dịch gây quỹ

Để xem thông tin chi tiết của chiến dịch, người đóng góp có thể ấn chọn chiến dịch đó và sau đó là trang thông tin chi tiết chiến dịch như ở hình 4.13.



Hình 4.11: Giao diện người dùng khi xem thông tin chiến dịch đang chờ xét duyệt



Hình 4.12: Giao diện của trang danh sách các chiến dịch

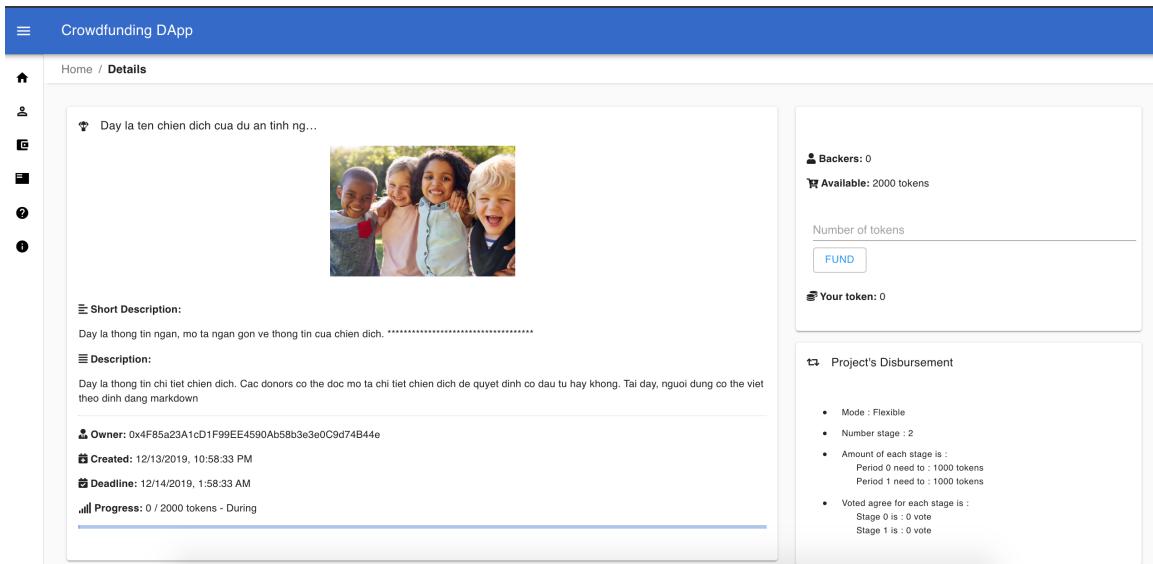
4.2 Đánh giá hệ thống đã hiện thực

4.2.1 Kiểm tra các quy trình trong hệ thống

4.2.1.1 Mục tiêu

Kiểm tra các quy trình trong hệ thống nhằm mục tiêu sau:

- Xác định đầu ra các hàm trong hợp đồng thông minh có trả về kết quả đúng như mong đợi với đầu vào cho trước hay không.



Hình 4.13: Giao diện của trang chi tiết thông tin chiến dịch

- Việc thực hiện kiểm tra các hàm tuần tự như quy trình hoạt động thực tế của hệ thống.
- Phát hiện các lỗi ở mức đơn vị nhỏ nhất trong hệ thống.

4.2.1.2 Phương pháp thực hiện

Nhóm tác giả tiến hành kiểm thử đơn vị (unit testing) với các hàm trong hệ thống theo hai kịch bản. Bao gồm:

- **Kịch bản 1:** kiểm thử quy trình với chiến dịch một giai đoạn.
- **Kịch bản 2:** kiểm thử quy trình với chiến dịch nhiều giai đoạn.

Kịch bản 1 được nhóm tác giả soạn như sau:

- (1) Nhân viên vận hành tiến hành biên dịch và triển khai các hợp đồng thông minh lên mạng blockchain. Kiểm tra kết quả: địa chỉ các hợp đồng thông minh đã có trên mạng blockchain hay chưa?
- (2) Người đóng góp gọi hàm *deposit* để nộp tiền vào hệ thống. Kiểm tra kết quả: chênh lệch số dư trước và sau khi gọi hàm có bằng số tiền đã nộp?
- (3) Người vận hành gọi hàm *addVerify* để thêm danh sách xác nhận viên xác minh. Kiểm tra kết quả: địa chỉ có trong danh sách địa chỉ nhân viên xác minh?
- (4) Người tạo chiến dịch gọi hàm *registerIdentity* để đăng ký hồ sơ định danh. Kiểm tra kết quả: lấy thông tin hồ sơ định danh đã lưu đổi chiều với đầu vào trước đó đã nhập, hai thông tin có giống nhau?
- (5) Nhân viên xác minh gọi hàm *verify* để xác minh cho một hồ sơ định danh. Kiểm tra

kết quả: trạng thái của hồ sơ định danh đã được xác minh hay chưa?

- (6) Người tạo chiến dịch gọi hàm *createCampaign* để tạo chiến dịch thứ nhất. Kiểm tra kết quả: đối chiếu thông tin chiến dịch đã lưu với đầu vào đã nhập có khớp nhau?
- (7) Nhân viên xác minh gọi hàm *verifyCampaign* để xác minh cho chiến dịch, cho phép chiến dịch được nhận đóng góp từ người dùng. Kiểm tra kết quả: trạng thái chiến dịch có được xác minh hay chưa?
- (8) Người đóng góp gọi hàm *donate* để đóng góp tiền cho chiến dịch, số tiền đóng góp đúng bằng mục tiêu của chiến dịch. Kiểm tra kết quả: chênh lệch số dư trước và sau khi đóng góp có bằng với số tiền đã đóng góp?
- (9) Người tạo chiến dịch tiến hành gọi hàm *endCampaign* để gọi lệnh giải ngân từ chiến dịch khi kết thúc thời gian chiến dịch gây quỹ. Kiểm tra kết quả: chênh lệch số dư (tokens) của người tạo chiến dịch trước và sau khi gọi lệnh giải ngân đúng bằng số tiền đã đóng góp hay không?
- (10) Người tạo chiến dịch gọi hàm *createCampaign* để tạo chiến dịch thứ hai. Kiểm tra kết quả: đối chiếu thông tin chiến dịch đã lưu với đầu vào đã nhập có khớp nhau?
- (11) Nhân viên xác minh gọi hàm *verifyCampaign* để xác minh cho chiến dịch, cho phép chiến dịch được nhận đóng góp từ người dùng. Kiểm tra kết quả: trạng thái chiến dịch có được xác minh hay chưa?
- (12) Người đóng góp gọi hàm *donate* để đóng góp tiền cho chiến dịch, số tiền đóng góp không đạt được mục tiêu gây quỹ. Kiểm tra kết quả: chênh lệch số dư trước và sau khi đóng góp có bằng với số tiền đã đóng góp?
- (13) Sau khi kết thúc thời gian gây quỹ, người đóng góp tiến hành kiểm tra số dư có đúng bằng số tiền đã đóng góp hay không?

Kịch bản 2 như sau:

- (1) Nhân viên vận hành tiến hành biên dịch và triển khai các hợp đồng thông minh lên mạng blockchain. Kiểm tra kết quả: địa chỉ các hợp đồng thông minh đã có trên mạng blockchain hay chưa?
- (2) Người đóng góp gọi hàm *deposit* để nộp tiền vào hệ thống, số lượng tài khoản người đóng góp là 5 và mỗi tài khoản sẽ có 1500 tokens dùng cho đóng góp chiến dịch. Kiểm tra kết quả: chênh lệch số dư trước và sau khi gọi hàm có bằng số tiền đã nộp?
- (3) Người vận hành gọi hàm *addVerify* để thêm danh sách xác nhân viên xác minh. Kiểm tra kết quả: địa chỉ có trong danh sách địa chỉ nhân viên xác minh?

- (4) Người tạo chiến dịch gọi hàm *registerIdentity* để đăng ký hồ sơ định danh. Kiểm tra kết quả: lấy thông tin hồ sơ định danh đã lưu đối chiếu với đầu vào trước đó đã nhập, hai thông tin có giống nhau?
- (5) Nhân viên xác minh gọi hàm *verify* để xác minh cho một hồ sơ định danh. Kiểm tra kết quả: trạng thái của hồ sơ định danh đã được xác minh hay chưa?
- (6) Người tạo chiến dịch gọi hàm *createCampaign* để tạo chiến dịch thứ nhất, chiến dịch thứ nhất có 3 giai đoạn giải ngân. Kiểm tra kết quả: đối chiếu thông tin chiến dịch đã lưu với đầu vào đã nhập có khớp nhau?
- (7) Nhân viên xác minh gọi hàm *verifyCampaign* để xác minh cho chiến dịch, cho phép chiến dịch được nhận đóng góp từ người dùng. Kiểm tra kết quả: trạng thái chiến dịch có được xác minh hay chưa?
- (8) Người đóng góp gọi hàm *donate* để đóng góp tiền cho chiến dịch, số tiền đóng góp đúng bằng mục tiêu của chiến dịch. Kiểm tra kết quả: chênh lệch số dư trước và sau khi đóng góp có bằng với số tiền đã đóng góp?
- (9) Người tạo chiến dịch tiến hành gọi hàm *endCampaign* để gọi lệnh giải ngân giai đoạn đầu tiên từ chiến dịch khi kết thúc thời gian chiến dịch gây quỹ. Kiểm tra kết quả: chênh lệch số dư (tokens) của người tạo chiến dịch trước và sau khi gọi lệnh giải ngân đúng bằng số tiền được giải ngân ở giai đoạn đầu tiên hay không?
- (10) Người đóng góp tiến hành gọi hàm *vote* để bình chọn cho giai đoạn thứ hai trở đi. Kiểm tra kết quả: danh sách đã bình chọn cho giai đoạn giải ngân của chiến dịch có địa chỉ của người đóng góp hay không?
- (11) Người tạo chiến dịch tiến hành gọi hàm *endCampaign* lần nữa để gọi lệnh giải ngân giai đoạn tiếp theo. Kiểm tra kết quả: chênh lệch số dư (tokens) của người tạo chiến dịch trước và sau khi gọi lệnh giải ngân đúng bằng số tiền được giải ngân ở giai đoạn đó hay không?

Với kịch bản 2, nhóm tác giả tiến hành tạo ra nhiều chiến dịch khác nhau, sau đó người đóng góp tiến hành biểu quyết với tỉ lệ đồng ý khác nhau. Kịch bản được nhóm tác giả hiện thực bằng ngôn ngữ Nodejs đính kèm với khóa luận này.

4.2.1.3 Kết quả thực hiện

Kết quả chạy kịch bản kiểm thử với kịch bản 1 được thể hiện ở hình 4.14.

Hình 4.15 là kết quả kiểm thử với kịch bản 2.

Contract: Campaign - one stage disbursement	
✓ Deposit wallet: 2 accounts with 1000 tokens / account	(360ms)
✓ Add verifier (138ms)	
✓ Register identity (226ms)	
✓ Verify an identity (134ms)	
✓ Create first campaign (243ms)	
✓ Accept first campaign (175ms)	
✓ Donate first campaign (reach target token) (596ms)	
✓ Disburse first campaign (1427ms)	
✓ Withdraw from Token to ETH (175ms)	
✓ Create second campaign (failed campaign) (250ms)	
✓ Accept second campaign (170ms)	
✓ Donate second campaign (NOT reach target token) (779ms)	
✓ Checking auto refund (1515ms)	

Hình 4.14: Kết quả kiểm thử kịch bản 1

Contract: Campaign - multi stage disbursement	
✓ Deposit: 5 accounts with 1500 tokens / account	(1060ms)
✓ Add verifier (218ms)	
✓ Register identity (280ms)	
✓ Verify an identity (134ms)	
✓ Create first campaign: multi stages (MODE 1) (384ms)	
✓ Accept first campaign (191ms)	
✓ Donate to first campaign (1519ms)	
✓ 1st campaign: Withdraw stage 0 (694ms)	
✓ 1st campaign: Voting for stage 1 (3/5 agree) (787ms)	
✓ 1st campaign: Withdraw stage 1 (373ms)	
✓ 1st campaign: Checking withdraw stage 2 without voting (184ms)	
✓ 1st campaign: Voting for stage 2 (2/5 agree) (638ms)	
✓ 1st campaign: Withdraw stage 2 -> fail (178ms)	
✓ Create second campaign: multi stages (MODE 2) (365ms)	
✓ Accept second campaign (182ms)	
✓ Donate to second campaign (1883ms)	
✓ 2nd campaign: Withdraw stage 0 (436ms)	
✓ 2nd campaign: Voting for stage 1 (2/5 agree) (631ms)	
✓ 2nd campaign: Withdraw stage 1 -> fail (177ms)	
✓ 2nd campaign: Checking withdraw stage 2 without voting (171ms)	
✓ 2nd campaign: Voting for stage 2 (5/5 agree) (701ms)	
✓ 2nd campaign: Withdraw stage 2 -> fail (181ms)	
✓ Create third campaign: multi stages (MODE 3) (377ms)	
✓ Accept third campaign (285ms)	
✓ Donate to third campaign (2007ms)	
✓ 3rd campaign: Withdraw stage 0 (423ms)	
✓ 3rd campaign: Voting for stage 1 => should error before time (142ms)	
✓ 3rd campaign: Voting for stage 1 (5/5 agree) (3122ms)	
✓ 3rd campaign: Withdraw stage 1 (475ms)	
✓ 3rd campaign: Voting for stage 2 => should error before time (152ms)	
✓ 3rd campaign: Voting for stage 2 (2/5 agree) (1351ms)	
✓ 3rd campaign: Withdraw stage 2 -> fail (189ms)	
✓ Create fourth campaign multi stages (MODE 4) (395ms)	
✓ Accept fourth campaign (180ms)	
✓ Donate to fourth campaign (2226ms)	
✓ 4th campaign: Withdraw stage 0 (678ms)	
✓ 4th campaign: Voting for stage 1 => should error before time (143ms)	
✓ 4th campaign: Voting for stage 1 (4/5 agree) (3158ms)	
✓ 4th campaign: Withdraw stage 1 (378ms)	
✓ 4th campaign: Voting for stage 2 => should error before time (134ms)	
✓ 4th campaign: Voting for stage 2 (2/5 agree) (2423ms)	
✓ 4th campaign: Withdraw stage 2 -> fail (188ms)	
✓ 4th campaign: Voting for stage 3 (5/5 agree) (1994ms)	
✓ 4th campaign: Withdraw stage 3 -> fail (207ms)	
✓ Create fiveth campaign: multi stages (MODE 1 - check again) (365ms)	
✓ Accept fiveth campaign (166ms)	
✓ Donate to fiveth campaign (2305ms)	
✓ 5th campaign: Withdraw stage 0 (546ms)	
✓ 5th campaign: Voting for stage 1 (3/5 agree but low ratio token) (644ms)	

Hình 4.15: Kết quả kiểm thử kịch bản 2

4.2.2 Đo lường tốc độ thực hiện giao dịch

Để tăng tính tin cậy cho phần đánh giá dưới đây của khóa luận, nhóm tác giả đã tham khảo mô hình đánh giá và kết quả đánh giá về hiệu suất của ethereum ở các công trình khác để làm thước đo và thực hiện tương tự với mô hình đánh giá đó.

Cụ thể, công trình của tác giả Sara Rouhani và Ralph Deters **rouhani2017performance** đã đo được thời gian trung bình cho mỗi transaction là 104.609ms với Parity client và 198.9125ms với Geth. Tổng số transaction được gửi là 2000. Hai Ethereum private blockchain khác nhau với cùng cấu hình được thực thi bởi Parity client và Geth client được sử dụng để đo lường. Cấu hình hệ thống bao gồm 24GB RAM và Core i7-6700 CPU. Việc gửi các transaction được thực hiện bằng ngôn ngữ NodeJS và sau đó thu thập thời gian xử lý cho việc xác nhận các transaction.

4.2.2.1 Môi trường thực hiện đánh giá

Nhóm tác giả thực hiện việc đánh giá này trên cấu hình máy như sau:

- **Chip xử lí:** 4 x Intel(E) Pentium(R) CPU N3540 @ 2.16GHz
- **RAM:** 8GB
- **Hệ điều hành:** Alpine Linux 3.9 chạy trên Docker container

Công cụ mà nhóm tác giả sử dụng trong phần đánh giá này là **Truffle framework**¹⁹, đây là một bộ công cụ được sử dụng để triển khai các hợp đồng thông minh hỗ trợ ngôn ngữ Solidity.

Trong phần đánh giá thời gian này, nhóm tác giả sử dụng một mạng ethereum riêng được chạy trên mạng cục bộ nhằm loại bỏ đi thời gian chờ xác nhận giao dịch thông thường trên các mạng công khai hiện tại.

4.2.2.2 Phương pháp thực hiện đánh giá

Đầu tiên nhóm tác giả thực hiện lựa chọn các hàm trong hợp đồng thông minh thường xuyên được sử dụng trong hệ thống, sau đó các hàm được chọn sẽ được hiện thực thông qua các transaction. Các transaction sẽ được xử lý và gửi đi bằng NodeJS, thời gian đo được tính từ lúc transaction được tạo ra đến lúc hoàn tất transaction đó. Với mỗi transaction, thực hiện gửi đi tuần tự 100, 200, 400, 600, 900 lần với cùng một bộ tham số cho trước. Sau đó lấy kết quả là thời gian trung bình thực hiện cho mỗi transaction.

Các hàm được chọn và tham số đầu vào cho mỗi hàm để đo thời gian được thể hiện ở bảng 4.1.

Contract	Hàm được chọn	Tham số đầu vào	Ghi chú
Wallet	deposit()		
Campaigns	createCampaign()	77760000, 1000000, 1, [], 0, [], '8f1ef45972ebd8ef45b2410e8a0b399181fed3d929738d2eb96baf470758a97d', 'c2337a3217ffcf3b01398d83577a1c32235ceb4f481b8c7be00a055798e95d36'	một g.đoạn giải ngân
	createCampaign()	77760000, 1000000, 3, [300000, 300000, 400000] 2, [0, 7200, 7200], '8f1ef45972ebd8ef45b2410e8a0b399181fed3d929738d2eb96baf470758a97d', 'c2337a3217ffcf3b01398d83577a1c32235ceb4f481b8c7be00a055798e95d36'	nhiều g.đoạn giải ngân
	donate()	0, 1	

Bảng 4.1: Các hàm và tham số đầu vào được dùng để đo thời gian thực hiện giao dịch

¹⁹<https://www.trufflesuite.com>

4.2.2.3 Kết quả đánh giá

Sau khi thực hiện đánh giá thời gian, nhóm tác giả đã tổng hợp kết quả như ở bảng 4.2. Theo như kết quả tổng hợp được, nhóm tác giả nhận xét rằng với dữ liệu đầu vào càng nhiều (kích thước lớn) thì thời gian xử lý càng lâu.

Contract	Hàm	Thời gian thực hiện 100 lần (giây)	Thời gian thực hiện 200 lần (giây)	Thời gian thực hiện 400 lần (giây)	Thời gian thực hiện 600 lần (giây)	Thời gian thực hiện 900 lần (giây)	Thời gian trung bình (giây)	Ghi chú
Wallet	deposit	11.099	18.035	36.484	54.166	86.968	0.095856	
Campaigns	createCampaign	19.613	37.839	76.577	115.433	172.447	0.1921527	một g.đoạn giải ngắn
	createCampaign	31.986	63.561	127.236	191.94	287.694	0.319063	nhiều g.đoạn giải ngắn
	donate	16.529	32.083	63.9	94.204	142.934	0.160255	

Bảng 4.2: Bảng kết quả đánh giá thời gian hiện thực một số hàm trong hệ thống.

Từ kết quả thu được, ta có biểu đồ tổng quan về tốc độ thực hiện của các hàm như ở hình 4.16.

Nhìn vào kết quả, ta thấy được:

- Hàm *deposit* là hàm có tốc độ thực hiện nhanh nhất với thời gian trung bình là 0.095856 giây. Phần mã xử lí của hàm *deposit* tương đối ngắn nên thời gian xử lí nhanh hơn.
- Hàm có tốc độ xử lí chậm nhất là hàm *createCampaign* (với chiến dịch gây quỹ có nhiều giai đoạn giải ngắn). Do hàm này có tham số đầu vào tương đối nhiều và phần mã xử lí phức tạp hơn nên thời gian hoàn tất lâu hơn những hàm khác.

4.2.3 Chi phí thực hiện các giao dịch trong hệ thống

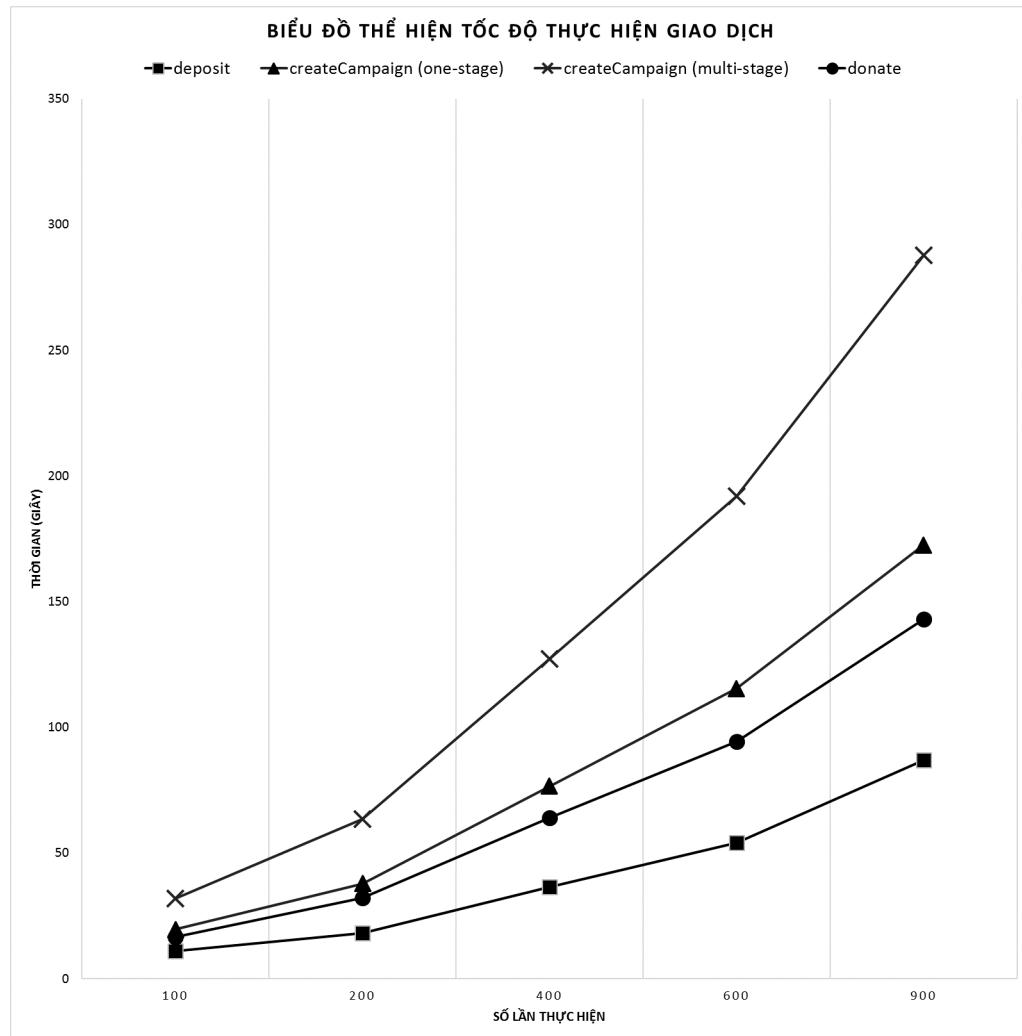
4.2.3.1 Môi trường thực hiện đánh giá

Nhóm tác giả thực hiện việc đo lường chi phí giao dịch trên cấu hình máy như sau:

- **Chip xử lí:** 4 x Intel(E) Pentium(R) CPU N3540 @ 2.16GHz
- **RAM:** 8GB
- **Hệ điều hành:** Alpine Linux 3.9 running in Docker container

Bộ công cụ **Truffle framework** kết hợp plug-in có tên là **eth-gas-reporter**²⁰ được sử dụng để triển khai các hợp đồng thông minh và đo lường chi phí thực hiện. Mạng ethereum riêng chạy trên máy cục bộ được sử dụng nhằm loại bỏ đi thời gian chờ xác nhận giao dịch thông thường trên các mạng công khai hiện tại.

²⁰<https://www.npmjs.com/package/eth-gas-reporter>



Hình 4.16: Biểu đồ thể hiện tốc độ thực hiện giao dịch giữa các hàm

4.2.3.2 Phương pháp thực hiện đánh giá

Do chỉ có các hàm thực hiện ghi dữ liệu mới tôn chi phí thực hiện nên nhóm tác giả chọn ra các hàm có thao tác ghi dữ liệu, sau đó các hàm được chọn sẽ được hiện thực thông qua các transaction. Các transaction sẽ được xử lý và gửi đi bằng NodeJS. Sau đó thực hiện ghi lại kết quả chi phí.

Các hàm được chọn và tham số đầu vào cho mỗi hàm để đo lường chi phí được thể hiện ở bảng 4.3. Các tham số đầu vào mẫu được cho là sát với thực tế khi triển khai hệ thống (độ dài từng tham số mẫu là sát với thực tế).

4.2.3.3 Kết quả đo lường chi phí các giao dịch trong hệ thống

Bảng tổng hợp chi phí cho các giao dịch được liệt kê ở bảng 4.4. Kết quả màn hình khi chạy công cụ đo lường chi phí được thể hiện ở hình 4.17.

Contract	Hàm	Dữ liệu đầu vào	Ghi chú
Wallet	deposit		
	withdraw	1000	
Identity	addVerify	'0x93598a39777ED4B4Af3Ac7429d123Ca3bE9658C5', 'AAAAB3NzaC1yc2EAAAQABAAAGQCDxb02O3XWhktz4Hwi6/61ltfk/SCqeXLufvjr6O3wh1++MmTzT+Kzc00azsKsiFJTXL7ynC06Vp1Hp9o0BK3Q/QZTo8jRoP3XX1LBu1CLe7OeOA5P2TO/nz2mWtuxz0b11GmRrjO8YoznizlPiolLkv9hoDBvwTy0JonyJ6+w=='	
	changePubKey	'0x93598a39777ED4B4Af3Ac7429d123Ca3bE9658C5', 'MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCMjs5j52lzXN6XX+nZ1jsyaBgzVBsA/JIWVux1zL0pw4GocvqPsZrIKwKsTeQycGdf3azjKRKwMga6g8fPFHO+Ayh+6v33B1h+3ckWu81alwsM+Y9ADpcMret5qh2Mv9rDyWi+lmAYeUAOOosAWfmgc6QJz+psSMtuGKOr08q+1wIDAQAB'	
	registerIdentity	'KLTN', 'UIT-HCM, Linh Trung, Thu Duc, HCM', 830550240, 'QmarHSr9aSNaPSR6G9KFPbuLV9aEqJfTk1y9B8pdwqK4Rq', 'frPULs0boASMCqSq1igu+jX636wkY+fzhFSRnFQi9dQuK50yzCobUIGm5b/f7oGDea/NrieB5c883EpWiQdgJlO+0B43jLAtfSJ/mlbGX3FUPc6LAQzxIcb5FSh7+Q1E4WIUyFwLwoNdipDYFcpuXx1CsKeepjFHWGFhfupxM=', '0x93598a39777ED4B4Af3Ac7429d123Ca3bE9658C5'	
	verify	'0x41A418C946Fd3201b7b2b30B367De35b0c54A6ce', true	
Campaigns	createCampaign	7776000, 1000000, 1, [], 0, [], '8f1ef45972ebd8ef45b2410e8a0b399181fed3d929738d2eb96ba470758a97d', 'c2337a3217ffcf3b01398d83577a1c32235ceb4f481b8c7be00a055798e95d36'	một g.đoạn giải ngắn
	createCampaign	10, 1000, 3, [300, 300, 400], 0, [], '8f1ef45972ebd8ef45b2410e8a0b399181fed3d929738d2eb96ba470758a97d', 'c2337a3217ffcf3b01398d83577a1c32235ceb4f481b8c7be00a055798e95d36'	nhiều g.đoạn giải ngắn
	verifyCampaign	1, true	
	donate	1, 1000	
	claimRefund	1, 200	
	donate	1, 200	donate lần 2
Disbursement	endCampaign	1	
Disbursement	vote	1, 1, true	

Bảng 4.3: Các hàm và dữ liệu đầu vào được dùng để đo lường chi phí giao dịch

Công cụ đo lường còn cung cấp cho chúng ta chi phí triển khai các hợp đồng thông minh, kết quả được thể hiện ở bảng 4.5.

Một số đại lượng trong bảng kết quả:

- **Gas** - là một đơn vị đo lường công việc tính toán của các giao dịch hoặc hợp đồng thông minh trong mạng Ethereum.
- **ETH** - là một đơn vị tiền tệ được sử dụng nội bộ trong mạng Ethereum. Tỉ lệ trao

```

Contract: Perform cost all functions
✓ contract Wallet: function deposit (64512 gas)
✓ contract Wallet: function withdraw (46257 gas)
✓ contract Identity: function addVerifier (265666 gas)
✓ contract Identity: function changePubKey (79644 gas)
✓ contract Identity: function registerIdentity (430215 gas)
✓ contract Identity: function verify (20778 gas)
✓ contract Campaigns: function createCampaign (ONE-STAGE) (281589 gas)
✓ contract Campaigns: function createCampaign (MULTI-STAGE) (497048 gas)
✓ contract Campaigns: function verifyCampaign (46288 gas)
✓ contract Campaigns: function donate (177817 gas)
✓ contract Campaigns: function claimRefund (36007 gas)
✓ contract Campaigns: function donate (again) (48473 gas)
✓ contract Campaigns: function endCampaign (82927 gas)
✓ contract Disbursement: function vote (91003 gas)

```

Hình 4.17: Ảnh chụp màn hình kết quả đo lường chi phí giao dịch

đổi giữa các đại lượng như sau: giá gas là 2 GWei/gas, và $1 \text{ ETH} = 10^9 \text{ GWei}$. Giá trị chuyển đổi giữa ETH và USD hiện tại được tham khảo trên **CoinMarketcap**²¹ là 150.08 USD/ETH (cập nhật ngày 27/11/2019)

Contract	Hàm	Chi phí tính toán (gas)	Chi phí giao dịch (ETH)	Chi phí giao dịch (USD)
Wallet	deposit	64512	0.000129024	0.02
	withdraw	46257	0.000092514	0.01
Identity	addVerify	265666	0.000531332	0.08
	changePubKey	79644	0.000159288	0.02
	registerIdentity	430215	0.000860430	0.13
	verify	20778	0.000041556	0.01
Campaigns	createCampaign	281589	0.000563178	0.08
	createCampaign	497048	0.000994096	0.15
	verifyCampaign	46288	0.000092576	0.01
	donate	177817	0.000355634	0.05
	claimRefund	36007	0.000072014	0.01
	donate	48473	0.000096946	0.01
	endCampaign	82927	0.000165854	0.02
Disbursement	vote	91003	0.000182006	0.03

Bảng 4.4: Kết quả đo lường chi phí giao dịch

Đánh giá tổng quan về chi phí:

- Hàm *verify* trong contract Identity có mức chi phí thực hiện thấp nhất với 20778 gas (tương đương 0.000041556 ETH). Phần mã xử lí của hàm này tương đối ngắn nên chi phí thấp hơn.
- Hàm có chi phí cao nhất là hàm *createCampaign* (tạo chiến dịch với nhiều giai đoạn)

²¹<https://coinmarketcap.com>

Contract	Chi phí (gas)	Chi phí (ETH)	Chi phí (USD)
Campaigns	3447461	0.006894922	1.03
Disbursement	1331555	0.002663110	0.4
Identity	2401480	0.004802960	0.72
Wallet	1163284	0.002326568	0.35

Bảng 4.5: Kết quả đo lường chi phí triển khai các hợp đồng

với chi phí 497048 gas (tương đương 0.000994096 ETH). Do hàm này có tham số đầu vào tương đối nhiều và phần mã xử lí phức tạp hơn nên chi phí cao hơn những hàm khác.

- Contract *Campaigns* có chi phí triển khai cao nhất (3447461 gas), contract có chi phí triển khai thấp nhất là *Identity* với 1163284 gas.

4.2.4 Phân tích bảo mật của hợp đồng thông minh trong hệ thống

4.2.4.1 Các lỗ hổng phổ biến trong hợp đồng thông minh trên Ethereum

Các lỗ hổng phổ biến được nhóm tác giả tham khảo từ công trình khảo sát của tác giả Nicola Atzei **atzei2016survey**, bảng 4.6 là bảng tổng hợp được trình bày dưới đây.

Level	Cause of vulnerability
Solidity	Call to the unknown
	Gasless send
	Exception disorders
	Type casts
	Reentrancy
	Keeping secrets
EVM	Immutable bugs
	Ether lost in transfer
	Stack size limit
Blockchain	Unpredictable state
	Generating randomness
	Time constraints

Bảng 4.6: Bảng tổng hợp các lỗ hổng trong hợp đồng thông minh trên Ethereum **atzei2016survey**.

Phần này chỉ tập trung vào giới thiệu các lỗi phổ biến với smart contract được lập trình bằng ngôn ngữ Solidity.

Call to unknown – một lỗ hổng khi gọi tới một vài lệnh cơ bản trong solidity như `call`, `send`, `delegatecall`. Nếu đối tượng được gọi trong các hàm này không tồn tại thì sẽ có

một tác dụng phụ là gọi đến hàm fallback²² của người nhận.

Gasless send – khi sử dụng hàm `send` để chuyển ether sang contract, có thể xảy ra lỗi `out-of-gas` (hết gas / không đủ gas) do việc thực thi hàm `send` đến contract sẽ dẫn đến gọi fallback của contract, mà số gas giới hạn trong trường hợp này là 2300 gas. Mà 2300 gas chỉ cho phép thực hiện một tập bytecode giới hạn, ví dụ: những lệnh / hàm không làm thay đổi trạng thái của contract. Nếu fallback có những tập lệnh phức tạp sẽ dẫn đến lỗi, khi đó người dùng sẽ bị mất toàn bộ phí thực hiện transaction.

Exception disorder – có thể tạm dịch lỗi hổng này là làm rối exception. Trong smart contract có một điều đặc biệt là với một exception được ném ra thì sẽ có hai cách xử lí khác nhau phụ thuộc vào cách các contract gọi nhau. Cụ thể:

- Nếu tất cả lời gọi hàm trong chuỗi đều là lời gọi trực tiếp, thì việc thực thi dừng lại và mọi giá trị trong lúc thực thi được hoàn nguyên.
- Nếu có ít nhất một lời gọi hàm được thực hiện thông qua lệnh `call` (tương tự với `delegatecall` và `send`), thì exception được truyền dọc theo chuỗi, chỉ hoàn nguyên các giá trị thực thi bên trong hàm của lời gọi call. Từ thời điểm đó (từ lệnh `call`), việc thực hiện được nối lại, và lệnh `call` trả về `false`.

Để hiểu rõ về lỗi hổng, ta xem xét một ví dụ sau:

```
contract Alice { function ping(uint) returns (uint) }

contract Bob {
    uint x=0;
    function pong(Alice c) {
        x=1;
        c.ping(42);
        x=2;
    }
}
```

Có hai trường hợp xảy ra:

- Trường hợp 1: thực hiện gọi hàm `pong` của Bob và sau đó hàm `ping` của Alice ném ra một exception. Sau đó, việc thực thi dừng lại và các giá trị của toàn bộ giao dịch được hoàn nguyên. Do đó, biến `x` có giá trị là 0 sau khi kết thúc việc thực thi.
- Trường hợp 2: giả sử contract Bob gọi `ping` của Alice thông qua lệnh `call` và sau đó `ping` của Alice ném ra một exception. Khi đó, chỉ có giá trị bên trong hàm `ping` được hoàn nguyên và lúc này khi kết thúc việc thực thi thì giá trị của `x` là 2.

²²fall back là một hàm đặc biệt trong smart contract, là hàm không có tên và được sử dụng khi: contract nhận ether, hoặc khi có ai đó gọi hàm không có trong contract hoặc tham số không đúng.

Sự bất thường trong cách xử lý các trường hợp exception có thể ảnh hưởng đến tính bảo mật của contract. Chẳng hạn, sẽ khiến lập trình viên tin rằng việc chuyển ether thông qua hàm send là thành công chỉ vì không có exception nào có thể dẫn đến các cuộc tấn công.

Type casts – trình biên dịch Solidity có thể phát hiện một số lỗi kiểu dữ liệu. Ví dụ: gán một giá trị số nguyên cho một biến thuộc kiểu chuỗi. Khi thực hiện gọi một hàm hay chương trình con nào đó, người gọi phải khai báo interface của hàm hay chương trình con đó. Nhưng điều này chỉ kiểm tra được hàm đó có tồn tại không, chứ không xác định được mã thực thi của hàm đích có trùng khớp với hàm thực tế cần gọi hay không. Do đó, điều này có thể đánh lừa lập trình vì khi thực thi sẽ không có bất lỗi nào được ném ra dù cho việc gọi hàm không đúng như mong muốn.

Reentrancy – lỗ hổng này cho phép gọi lại một hàm không đệ quy lần nữa nhờ vào cơ chế gọi fallback của solidity. Như đã đề cập trước đó, nếu ta sử dụng lệnh call hoặc các lệnh tương tự mà đích đến không tồn tại hoặc rỗng thì sẽ thực hiện gọi fallback của contract, điều này có thể dẫn đến vòng lặp, vòng lặp này kết thúc khi tiêu thụ hết gas. Để rõ hơn lỗ hổng này, xem xét ví dụ sau:

```
contract Bob {
    bool sent = false;
    function ping(address c) {
        if (!sent) {
            c.call.value(2)();
            sent = true;
        }
    }
}

contract Attacker {
    function() {
        Bob(msg.sender).ping(this);
    }
}
```

Cách khai thác ví dụ trên:

- Chức năng của hàm ping trong contract **Bob**: gửi 2wei đến địa chỉ **c**, sử dụng lệnh **call** có đích đến rỗng (tức không gọi hàm nào) và không có giới hạn gas.
- Bây giờ, giả sử rằng ping đã được gọi với địa chỉ của **Attacker**. Như đã đề cập trước đó, **call** có tác dụng phụ là gọi fallback của **Attacker**, lần lượt gọi lại ping. Vì biến **sent** chưa được đặt thành **true**, Bob gửi lại 2wei cho Attacker và gọi lại fallback lần nữa, do đó bắt đầu một vòng lặp. Vòng lặp này kết thúc khi việc thực thi cuối cùng hết gas hoặc khi đạt đến giới hạn ngăn xếp hoặc khi Bob đã rút hết ether của mình.

Keeping secrets – trong smart contract hỗ trợ việc khai báo cấp độ truy xuất dữ liệu từ riêng

tư đến công khai. Tuy nhiên việc khai báo một biến hay một hàm nào đó là riêng tư chỉ có tác dụng ngăn người dùng hoặc contract khác gọi trực tiếp, trên thực tế người dùng vẫn đọc được giá trị của các biến được khai báo là riêng tư trong smart contract, do đó không đảm bảo được độ bí mật của dữ liệu. Điều này là do mỗi lời gọi hàm được đóng gói trong transaction, và các transaction này được công khai trong toàn mạng blockchain. Vì thế, mọi người đều có thể kiểm tra nội dung của giao dịch và suy ra giá trị của các trường dữ liệu.

4.2.4.2 Các công cụ phân tích bảo mật smart contract

Danh sách các công cụ phân tích, đánh giá smart contract giới thiệu bên dưới được tham khảo từ công trình của tác giả Jiaming Ye **8728953** và luận văn thạc sĩ của tác giả Ardit Dika **dika2017ethereum**.

Smartcheck²³ – là một công cụ phân tích tĩnh giúp phát hiện ra những lỗi hoặc cảnh báo trong smart contract được viết bằng Solidity. Công cụ này có các ưu điểm sau:

- Giao diện dễ sử dụng, kiểm tra được nhiều contract.
- Các lỗi hổng có thể phát hiện: *Non-strict comparison with zero, Hardcoded address, DoS by external contract, Reentrancy, Overflow and Underflow in Solidity, Redundant fallback function,*
- Ở mỗi lỗi được phát hiện, smartcheck cung cấp cụ thể số dòng bị lỗi và đánh dấu chúng.
- Bên cạnh việc phát hiện lỗi hổng, smartcheck còn cung cấp giải pháp khắc phục từng lỗi hổng đó.

Remix²⁴ – là một IDE dựa trên web để phát triển hợp đồng thông minh. Ngoài ra, Remix IDE còn được sử dụng như một công cụ phân tích mã bảo mật để kiểm tra các lỗi hổng có thể xảy ra. Remix IDE chỉ được sử dụng để phân tích mã nguồn Solidity. Remix IDE có khả năng phân tích gần như 100% các lỗi hổng hiện tại của hợp đồng thông minh **dika2017ethereum**.

Slither²⁵ – là framework phân tích tĩnh Solidity được viết bằng Python 3. Slither chạy bộ các trình phát hiện lỗi hổng, in thông tin trực quan về chi tiết hợp đồng và cung cấp API để dễ dàng viết các phân tích tùy chỉnh. Slither cho phép các nhà phát triển tìm ra các lỗi hổng, tăng cường khả năng đọc mã và nhanh chóng phân tích với những sự tùy chỉnh. Các chức năng và ưu điểm của Slither:

- Phát hiện các lỗi hổng trong Solidity.
- Xác định nơi xảy ra tình trạng lỗi trong mã nguồn.

²³<https://tool.smartdec.net>

²⁴<https://remix.ethereum.org>

²⁵<https://github.com/crytic/slither>

- Dễ dàng tích hợp với bộ công cụ Truffle.
- Tích hợp các trình báo cáo để biểu diễn thông tin contract.
- Hỗ trợ API để viết các trình phân tích tùy chỉnh bằng Python.
- Khả năng phân tích các hợp đồng được viết với phiên bản Solidity > 0.4
- Phân tích chính xác 99,9% tất cả mã Solidity công khai.
- Thời gian thực hiện trung bình dưới 1 giây mỗi contract.

Cả ba công cụ được giới thiệu ở trên đều là phần mềm mã nguồn mở.

4.2.4.3 Kết quả phân tích

Nhóm tác giả sử dụng các công cụ đã được giới thiệu ở mục 4.2.4.2 để thực hiện phân tích trên các smart contract của hệ thống. Cụ thể kết quả với từng công cụ như sau:

Kết quả phân tích với Remix IDE được thể hiện ở hình 4.18. Kết quả công cụ phân tích của Remix chỉ đưa ra 5 cảnh báo, cụ thể ở cảnh báo đầu tiên là về việc sử dụng lệnh send để chuyển ether, thì nhóm tác giả đã kiểm tra kĩ lưỡng các giá trị trước và sau lệnh send này nên hạn chế được các lỗ hổng như reentrancy.

Hình 4.19 là kết quả khi thực hiện phân tích với công cụ Slither. Kết quả phân tích cho thấy số contract thực hiện quét là 6 với 40 bộ nhận diện và có 3 vấn đề được tìm thấy. Các vấn đề quét được là cảnh báo lỗ hổng Reentrancy, tuy nhiên cảnh báo này không ảnh hưởng đến bảo mật của contract.

Công cụ Smartcheck được cung cấp dưới dạng giao diện web, do đó kết quả phân tích trên Smartcheck cũng được công khai²⁶. Hình 4.20 là ảnh chụp kết quả phân tích các hợp đồng thông minh trong hệ thống trên trang Smartcheck. Trong kết quả ta thấy bên cột bên phải là danh sách các lỗi, đọc qua các lỗi thì đây là những cảnh báo giúp cho lập trình hiệu quả hơn, không ảnh hưởng tới bảo mật của hợp đồng thông minh.

Đánh giá chung kết quả phân tích cho thấy, hợp đồng thông minh được hiện thực của hệ thống không gặp phải vấn đề bảo mật nghiêm trọng trên các công cụ đã phân tích.

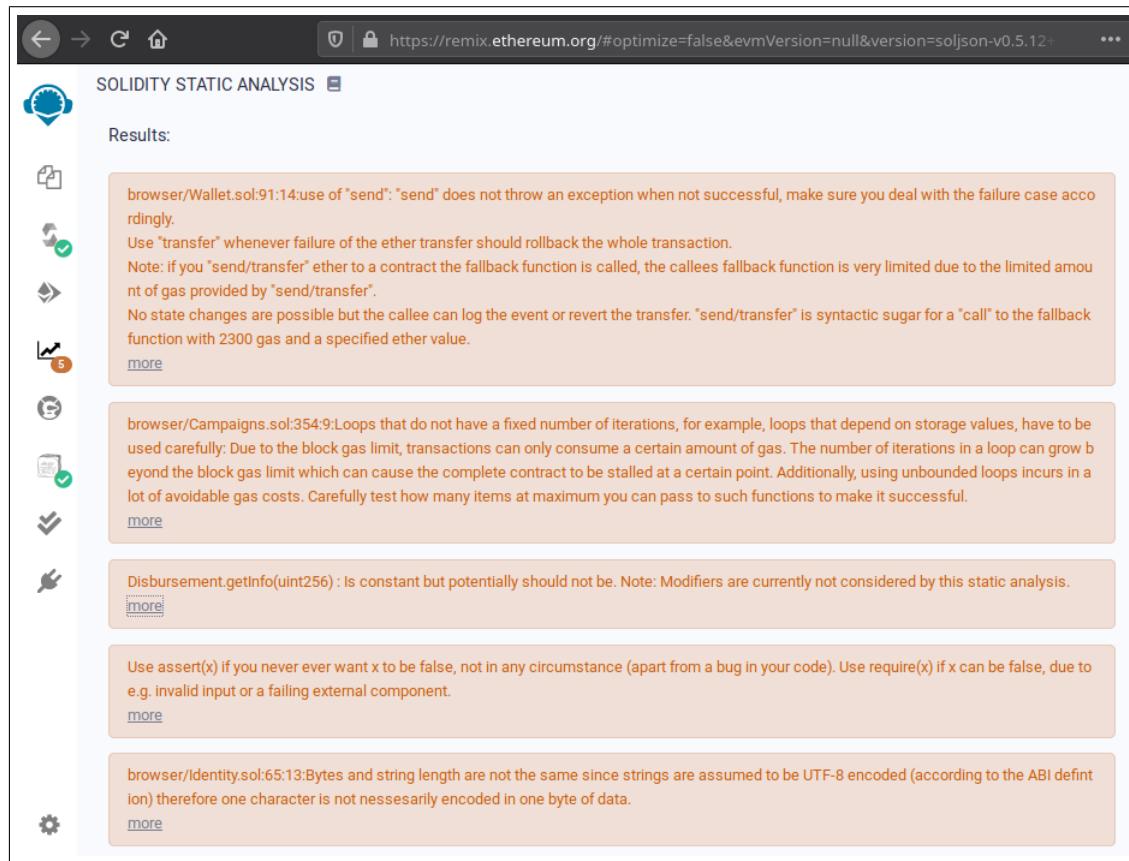
4.2.5 Đánh giá tốc độ tải trang của giao diện người dùng

4.2.5.1 Môi trường thực hiện

Thông tin cấu hình máy sử dụng để đo tốc độ truy xuất của ReactJS như sau:

- Chip xử lý: 2.7 Ghz Dual-Core intel Core i5
- Hệ điều hành: macOS Catalina version 10.15.1

²⁶Kết quả phân tích với Smartcheck: <https://tool.smartdec.net/scan/e6d6a5d37015403faf92395ec7f543c8>



Hình 4.18: Kết quả phân tích hợp đồng thông minh với Remix IDE

- RAM: 8GB 1867Mhz DDR3
- Đồ họa: Intel Iris Graphics 6100 1536 MB
- Trình duyệt Google Chrome version 78.0.3904 (được cài sẵn extension MetaMask đã cấp phép truy cập thông tin)

4.2.5.2 Kết quả đánh giá

Kết quả được tổng hợp ở bảng 4.7.

Page	Loading	Scripting	Rendering	Painting	System	Idle	Total
Detail campaign	28 ms	1320 ms	70 ms	65 ms	220 ms	918 ms	2621 ms
Home page	6 ms	743 ms	3 ms	3 ms	41 ms	102 ms	898 ms
Create campaign	24 ms	1181 ms	47 ms	36 ms	196 ms	672 ms	2156 ms
Explore campaigns	14 ms	798 ms	98 ms	82 ms	237 ms	1541 ms	2770 ms

Bảng 4.7: Bảng kết quả đo tốc độ truy xuất front-end của hệ thống

Đánh giá kết quả như sau:

- Trang danh sách chiến dịch (Explore campaign) có tổng thời gian hoàn thành lâu nhất với 2.77 giây, do trang này tải thông tin của nhiều chiến dịch, mỗi chiến dịch là một lời

```

> Compiling ./contracts/Campaigns.sol
> Compiling ./contracts/Disbursement.sol
> Compiling ./contracts/Identity.sol
> Compiling ./contracts/Migrations.sol
> Compiling ./contracts/SafeMath.sol
> Compiling ./contracts/Wallet.sol
> Artifacts written to /home/ethsec/app/build/contracts
> Compiled successfully using:
  - solc: 0.5.8+commit.23d335f2.Emscripten clang

- Fetching solc version list from solc-bin. Attempt #1

INFO:Detectors:
Reentrancy in Campaigns.createCampaign(uint256,uint256,uint256,uint256[],Disbursement.Mode,uint256[],string,string) (Campaigns.sol#112-200):
    External calls:
    - disb.create(campaigns.length - 1,numStage,amountStages,mode,times) (Campaigns.sol#190-196)
        Event emitted after the call(s):
        - Added(campaigns.length - 1) (Campaigns.sol#199)
Reentrancy in Campaigns.endCampaign(uint256) (Campaigns.sol#285-336):
    External calls:
    - ! token.addToken(msg.sender,amount) (Campaigns.sol#327)
        Event emitted after the call(s):
        - Paid(i,msg.sender,amount) (Campaigns.sol#326)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Reentrancy in Wallet.withdraw(uint256) (Wallet.sol#77-99):
    External calls:
    - ! msg.sender.send(weiValue) (Wallet.sol#91)
        Event emitted after the call(s):
        - Withdraw(msg.sender,weiValue) (Wallet.sol#90)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Slither:.. analyzed (6 contracts with 40 detectors), 3 result(s) found

```

Hình 4.19: Kết quả phân tích hợp đồng thông minh với Slither

gọi đến hợp đồng thông minh. Bên cạnh đó, mỗi chiến dịch cũng cần truy xuất thông tin đến cơ sở dữ liệu tập trung. Thật vậy vì tốc độ tải ban đầu (cột “loading”) tương đối thấp, do đó tổng thời gian hoàn thành trang lâu nhất là do ảnh hưởng các thành phần phụ thuộc kèm theo như đã đề cập.

- Trang chủ (Home page) có thời gian hoàn thành nhanh nhất với 0.898 giây, không lя gì do trang chủ hoàn toàn là một trang tinh.

The screenshot shows the SmartCheck web interface. On the left, there's a sidebar with a 'CONSULT' button and a 'Files' section containing 'Wallet.sol', 'Disbursement.sol' (which is highlighted in red), 'Campaigns.sol', and 'Identity.sol'. The main area displays the Solidity code for 'Disbursement.sol' with line numbers 13 to 31. The code includes comments and mappings. To the right, there's a 'KNOWLEDGE BASE' header, a user greeting 'HELLO, TUANLH2704@GMAIL.COM!', and navigation links 'START SCAN / LIST OF SCANS'. A vertical sidebar on the right lists 'Errors' and 'Lines' with specific findings: 'Costly loop', 'Compiler version not fixed', 'Revert inside the if-operator', 'Use of SafeMath', and 'Replace multiple return values with struct'.

```

13     uint[] agreed;
14     uint[] amountAgreed;
15     Mode mode;
16     mapping(address => mapping(uint => Vote)) voting; // user => stage => Vote
17 }
18
19 Campaigns internal camp;
20 mapping (uint => Data) internal stages;
21
22 /* -- Constructor -- */
23 //
24 /// @notice Constructor run only one time
25 /// @dev This contract MUST be run after TokenSystem
26 /// @param addrCampaign is address of Campaigns contract
27 constructor(Campaigns addrCampaign) public {
28     camp = addrCampaign;
29 }
30
31 /// @notice only some contracts MUST be run

```

Hình 4.20: Kết quả phân tích hợp đồng thông minh với Smartcheck

Chương 5

KẾT LUẬN

5.1 Kết quả đạt được

Khóa luận này đạt được một số kết quả sau:

- Đề xuất mô hình ứng dụng gây quỹ cộng đồng từ thiện dựa trên công nghệ blockchain cải thiện tính minh bạch bằng việc áp dụng hợp đồng thông minh, các điều khoản hợp đồng được thực thi tự động mà không có sự can thiệp từ con người. Các giao dịch đều được công khai trên mạng blockchain.
- Hiện thực mô hình đã đề xuất với các chức năng cơ bản của ứng dụng gây quỹ từ thiện cộng đồng như: tạo chiến dịch, đóng góp vào chiến dịch, giải ngân. Ngoài ra hệ thống còn hiện thực các chức năng nổi bật như: lưu trữ và quản lý thông tin định danh, hoàn tiền tự động khi chiến dịch không đạt được mục tiêu, giải ngân và bỏ phiếu giải ngân nhiều giai đoạn.
- Đánh giá hệ thống đã hiện thực với khía cạnh đánh giá như tốc độ thực hiện các giao dịch, chi phí thực hiện các giao dịch và bảo mật các hợp đồng thông minh.

5.2 Ưu điểm và khuyết điểm của hệ thống

5.2.1 Ưu điểm

Mô hình hệ thống đã hiện thực có các ưu điểm sau:

- Mô hình gây quỹ cộng đồng thông qua internet: tận dụng sức mạnh của cộng đồng để các chiến dịch gây quỹ từ thiện có thể lan tỏa tốt hơn đến nhiều người. Cũng chính cộng đồng sẽ là người giám sát các chiến dịch gây quỹ.
- Ứng dụng công nghệ blockchain và mô hình phi tập trung: các giao dịch được công

khai và minh bạch. Do đó tăng cường sự giám sát của cộng đồng với các chiến dịch. Mô hình phi tập trung loại bỏ sự can thiệp của bất cứ bên thứ ba nào làm thay đổi dữ liệu hệ thống.

- Cơ chế xác minh thông tin định danh và thông tin chiến dịch trước khi công khai đến cộng đồng: tuy yêu tố này sẽ giảm tính “phi tập trung” của ứng dụng nhưng mọi hành động về xét duyệt các thông tin đều được công khai cho cộng đồng.
- Tối ưu chi phí khi kết hợp lưu trữ thông tin trên cơ sở dữ liệu tập trung. Các giá trị liên quan đến tiền tệ tài chính được lưu trữ trên blockchain. Các thông tin về mô tả chiến dịch được lưu trữ trên cơ sở dữ liệu tập trung. Để đảm bảo tính toàn vẹn của dữ liệu, hệ thống đã kết hợp lưu trữ mã băm của dữ liệu lên blockchain.

5.2.2 Khuyết điểm

Một số khuyết điểm của hệ thống hiện tại được nhóm tác giả chỉ ra như sau:

- Chi phí và tốc độ thực hiện các giao dịch còn tương đối cao so với ứng dụng tập trung theo mô hình truyền thống. Đây là hạn chế của nền tảng blockchain công khai hiện tại. Với các giao dịch gọi tới các hàm có thay đổi dữ liệu trong hợp đồng thông minh đều tồn một chi phí nhất định. Xét về mặt kinh tế, thì đây là điểm hạn chế so với hệ thống theo mô hình tập trung. Còn xét về phương diện bảo mật thì đây là một ưu điểm có thể hạn chế các cuộc tấn công như spam, từ chối dịch vụ.
- Trong chức năng tự động hoàn tiền chiến dịch khi không đạt được mục tiêu gây quỹ, hàm kiểm tra số dư có sử dụng dấu thời gian hiện tại để xác định trạng thái của chiến dịch là kêu gọi thành công hay thất bại. Mà dấu thời gian này được xác định là thời gian của block mới nhất trong blockchain. Do đó, khi không có bất kì block nào được đóng vào thì trạng thái chiến dịch không được cập nhật và số dư cũng không được cập nhật thực tế. Việc cập nhật số dư có thể diễn ra chậm. Tuy nhiên sự sai lệch này là không đáng kể.
- Đối với chức năng giải ngân theo nhiều giai đoạn, khi một giai đoạn giải ngân không đạt đủ điều kiện về số phiếu đồng ý thì chưa có cơ chế hoàn tiền ở giai đoạn đó cho người đóng góp.

5.3 Khó khăn

Trong quá trình thực hiện khóa luận, nhóm tác giả gặp phải những khó khăn sau:

- Việc kiểm thử hệ thống chưa thể thực hiện chạy đồng thời nhiều giao dịch cùng lúc mà chỉ thực thi tuần tự các giao dịch.

- Nhóm tác giả chưa tìm được tài liệu về quy trình đánh giá, kiểm thử một ứng dụng blockchain chuẩn.
- Phần hiện thực hệ thống hiện tại chỉ có thể thực hiện trên mạng testnet và trên một nút mạng, chưa thực hiện trên mạng công khai nhiều nút mạng thực tế do vấn đề kinh phí triển khai. Nhưng việc triển khai hợp đồng thông minh trên một nút mạng thành công thì việc triển khai trên nhiều nút mạng là hoàn toàn có thể.

5.4 Hướng phát triển

Với những khuyết điểm và khó khăn đã đề ra, mục tiêu tiếp theo của khóa luận này như sau:

- Hoàn thiện chức năng hoàn tiền trong giải ngân nhiều giai đoạn.
- Hoàn thiện việc kiểm thử hệ thống với quy trình đánh giá chuẩn hơn.
- Hiện thực trên nhiều nền tảng và mạng blockchain khác nhau như EOS, HyperLedger, TomoChain,

Phụ lục A

Mã hợp đồng thông minh - Wallet

Mã nguồn hợp đồng thông minh Wallet được viết bằng ngôn ngữ Solidity:

```
pragma solidity ^0.5.3;
import {SafeMath} from "./SafeMath.sol";
import {Campaigns} from "./Campaigns.sol";

contract Wallet {
    using SafeMath for uint;

    Campaigns internal camp;
    uint internal mGranularity; //Minimum value of Wei
    uint internal mTotalBalances;
    address internal deployer;
    mapping(address => uint) internal mBalances; // wei

    event Deposit(address from, uint amount);
    event Withdraw(address to, uint amount);

    /* -- Constructor -- */
    //
    /// @notice Constructor to create a Wallet
    /// @dev This contract is deployed by system and only once deploy
    /// @param addrCampaign is address of Campaign contract
    constructor(Campaigns addrCampaign) public {
        camp = addrCampaign;
        deployer = msg.sender;
        mGranularity = 10**15; // 1 ETH = 1000 tokens
    }

    /// @dev granularity can be understood as the price of a token. 1 token =
    /// granularity form as wei
    /// @return the granularity of the token
    function granularity() external view returns (uint) { return mGranularity; }

    /// @dev Get token of user without campaigns
    /// @param user is address of user that you want check token
    /// @return Number of token
```

```

function balances(address user) external view returns(uint) {
    return mBalances[user] / mGranularity;
}

/// @notice get my balance (form as Token) of msg.sender with campaign
/// @param user is address of user
/// @return Result is number of token
function getBalance(address user) public view returns (uint) {
    if (mBalances[msg.sender] == 0) {
        return 0;
    }
    uint balance;
    balance = mBalances[msg.sender] / mGranularity;
    balance = balance.sub(camp.getAllDonation(user));
    return balance;
}

/// @notice Allow user transfer balances to this contract
/// @dev This function will receive balance that user send into contract and store
in contract
/// value will be stored in the mBalances variable
/// Amount is msg.value form as Wei (1 ETH = 10^18 wei)
function deposit() public payable {
    require(
        msg.value > 0,
        "Amount to deposit MUST be greater zero"
    );
    require(
        msg.value % mGranularity == 0,
        "Amount is not a multiple of granualrity"
    );
    mBalances[msg.sender] = mBalances[msg.sender].add(msg.value);
    mTotalBalances = mTotalBalances.add(msg.value);
    assert(address(this).balance >= mTotalBalances);
    emit Deposit(msg.sender, msg.value);
}

/// @notice This function allow user withdraw balances in contract to ETH
/// @dev Withdraw token in system to ETH. (Wei = token * mGranularity)
/// @param amount number of token that you want withdraw
/// @return `true` if withdraw process successful
function withdraw(uint amount) external returns (bool) {
    require(
        amount > 0,
        "Amount to deposit MUST be greater zero"
    );
    require(
        amount <= getBalance(msg.sender),
        "You don't have enough token"
    );
    uint weiValue = amount.mul(mGranularity); // exchange from token to Wei

    mBalances[msg.sender] -= weiValue;
    mTotalBalances = mTotalBalances.sub(weiValue);
    emit Withdraw(msg.sender, weiValue);
}

```

```

        if (!msg.sender.send(weiValue)) {
            mBalances[msg.sender] = mBalances[msg.sender].add(weiValue);
            mTotalBalances = mTotalBalances.add(weiValue);
            return false;
        } else {
            assert(address(this).balance >= mTotalBalances);
            return true;
        }
    }

    /// @notice Allow campaign owner (startups) can withdraw token from a succeed
    /// campaign
    /// @dev This function MUST be run by Campaign contract
    /// @param to is owner of campaign
    /// @param amount total token was sold in campaign
    /// @return `true` if withdraw process successful
    function addToken(address to, uint amount) external
    returns(bool)
    {
        require(
            address(camp) == msg.sender,
            "Sender address is invalid"
        );
        require(
            amount > 0,
            "Amount MUST be greater zero"
        );

        mBalances[to] = mBalances[to] + (amount * mGranularity);
        return true;
    }

    /// @notice This function for contract owner to change granularity
    /// @param newGranularity is new value of granularity
    function changeGranularity(uint newGranularity) external {
        require(
            msg.sender == deployer,
            "This function must be run by deployer"
        );
        mGranularity = newGranularity;
    }

    function () external payable {deposit();}
}

```

Phụ lục B

Mã hợp đồng thông minh - Campaigns

Mã nguồn hợp đồng thông minh Campaigns được viết bằng ngôn ngữ Solidity:

```
pragma solidity ^0.5.3;
import {SafeMath} from "./SafeMath.sol";
import {Wallet} from "./Wallet.sol";
import {Identity} from "./Identity.sol";
import {Disbursement} from "./Disbursement.sol";

contract Campaigns {
    Wallet internal token;
    Identity internal id;
    Disbursement internal disb;
    using SafeMath for uint;

    /* Explantation of campaign status
     * During: end date < now
     * Failed: end date >= now AND token collected < goal
     * Succeed: end date >= now AND token collected >= goal
     */
    enum Status {during, failed, succeed}

    /* Explantation of campaign FINACIAL status
     * Pending: new campaign just added. NOT allow donor fund to campaign
     * Accepted: a campaign was verified => Allow donors fund to campaign
     * Paid: a campaign that owner withdraw token completed => end campaign
     */
    enum FinStatus {pending, accepted, rejected, paid}

    struct CampaignInfo {
        address owner;
        uint startDate;
        uint endDate;
        uint goal;
        uint collected;
        uint stage;
        FinStatus finstt;
        string ref; // store reference to other info as name, description on db
    }
}
```

```

        string hashIntegrity; // hash of data store in server
        address[] donors;
        mapping(address => uint) donation;
        mapping(address => bool) isDonate;

    }

CampaignInfo[] internal campaigns;
mapping(address => uint[]) internal donor2campaigns; //mapping donors to campaigns
id
address internal deployer;
event Added(uint id);
event Accepted(uint id);
event Donated(uint id, address donor, uint token);
event Refund(uint id, address donor, uint token);
event Paid(uint id, address ownerCampaign, uint token);

/* -- Constructor -- */
//
/// @notice Constructor to create a campaign contract
/// @dev This contract MUST be run after Wallet
/// @param addrIdentity is address of Identity contract
constructor(Identity addrIdentity) public {
    deployer = msg.sender;
    id = addrIdentity;
}

/// @notice Update address of other contracts
/// @param addrWallet is address of Wallet contract
/// @param addrDisb is address of Disbursement contract
function linkOtherContracts(Wallet addrWallet, Disbursement addrDisb) external {
    require(
        msg.sender == deployer,
        "Only deployer"
    );
    token = addrWallet;
    disb = addrDisb;
}

/// @notice Get properties of a campaign
/// @param i is index of campaigns array
/// @return object {startDate, endDate, goal, collected, owner, finStatus, status, ref}
function getInfo(uint i) external view
returns(
    uint startDate,
    uint endDate,
    uint goal,
    uint collected,
    address owner,
    FinStatus finStatus,
    Status status,
    string memory ref,
    string memory hashIntegrity
) {

```

```

        ref = campaigns[i].ref;
        hashIntegrity = campaigns[i].hashIntegrity;
        startDate = campaigns[i].startDate;
        endDate = campaigns[i].endDate;
        goal = campaigns[i].goal;
        collected = campaigns[i].collected;
        owner = campaigns[i].owner;
        finStatus = campaigns[i].finstt;
        status = getStatus(i);
    }

    /// @notice Create a campaign
    /// @dev Add an element to variable campaigns array
    /// @param deadline is deadline for fundraising of a campaign. (unit: seconds)
    /// @param goal is goal of a campaign. Min-Max: 100.000-1.000.000.000
    /// @param numStage is number of stage withdraw campaign
    /// @param amountStages is amount of each stage withdraw campaign
    /// @param mode is mode for disburse campaign
    /// @param timeStages is deadline for each stage withdraw campaign
    /// @param ref is campaign reference to other information about campaign
    /// @param hashData is hash of data will store in db server, to check integrity
    function createCampaign(
        uint deadline,
        uint goal,
        uint numStage,
        uint[] calldata amountStages,
        Disbursement.Mode mode,
        uint[] calldata timeStages,
        string calldata ref,
        string calldata hashData
    )
    external {
        // To testing, you can comment following lines
        require(
            goal >= 1e5 && goal <= 1e9,
            "The goal of campaign must be include range is from 100.000 to 1.000.000.000
tokens"
        );
        require(
            id.isVerified(msg.sender),
            "You must be register identity and be accepted"
        );

        // To testing, you can comment following lines
        require(
            deadline >= 15 days,
            "The minimum fundraising time for the campaign is 15 days."
        );

        campaigns.push(CampaignInfo(
            msg.sender,
            now,
            now + deadline,
            goal,
            0,

```

```

        0,
        FinStatus.pending,
        ref,
        hashData,
        new address[](0)
    ));

    if (numStage > 1) {
        if (amountStages.length != numStage) {
            revert('number element amount stage is invalid');
        }
        if (mode >= Disbursement.Mode.TimingFlexible && timeStages.length != numStage) {
            revert('Number of deadline is invalid');
        }
        uint sumOfAmount;
        uint[] memory times = new uint[](numStage);
        for (uint i = 0; i < numStage; i++) {
            sumOfAmount += amountStages[i];
            if (mode >= Disbursement.Mode.TimingFlexible) {
                if (i == 0) {
                    times[0] = now + deadline;
                } else {
                    times[i] = times[i-1] + timeStages[i];
                }
            }
        }
        if (sumOfAmount != goal) {
            revert('Sum of amount must be equal goal');
        }
    }

    disb.create(
        campaigns.length-1,
        numStage,
        amountStages,
        mode,
        times
    );
}

emit Added(campaigns.length - 1);
}

/// @notice Count how many people donated into a campaign
/// @param i is index of campaign
/// @return Number of donors of a campaign
function getNumberOfDonors(uint i) external view returns(uint) {
    return campaigns[i].donors.length;
}

/// @notice Determine a campaign is allow all donor can invest to that campaign
/// @param i is index of campaigns array
/// @param isAccept is variable used for decide a campaign be allowed transact
function verifyCampaign(uint i, bool isAccept) external {
    require(

```

```

        id.isVerifier(msg.sender),
        "You MUST be verifier");
    campaigns[i].finstt = isAccept ? FinStatus.accepted : FinStatus.rejected;
    emit Accepted(i);
}

/// @notice Allow donor can donate to a campaign
/// @param i is index of campaigns array
/// @param amount is amount of token that you want to donate
function donate(uint i, uint amount) external {
    CampaignInfo memory campaign = campaigns[i];
    require(
        amount > 0,
        "amount of token must be greater than zero"
    );
    require(
        now <= campaign.endDate,
        "Campaign is ended"
    );
    require(
        campaign.collected < campaign.goal,
        "Campaign is reached goal"
    );
    require(
        campaign.collected + amount <= campaign.goal,
        "Amount without goal of campaign"
    );
    require(
        campaigns[i].finstt == FinStatus.accepted,
        "This campaign MUST be accepted and NOT paid"
    );
    require(
        amount <= (token.balances(msg.sender) - getAllDonation(msg.sender)),
        "You don't have enough token");

    campaigns[i].donation[msg.sender] = campaigns[i].donation[msg.sender].add(amount
);
    if (!campaigns[i].isDonate[msg.sender]) {
        donor2campaigns[msg.sender].push(i);
        campaigns[i].isDonate[msg.sender] = true;
        campaigns[i].donors.push(msg.sender);
    }
    campaigns[i].collected = campaigns[i].collected.add(amount);
    emit Donated(i, msg.sender, amount);
}

/// @notice Allow donor can claim refund when campaign during
/// @param i is index of campaigns array
/// @param amount Amount donor want withdraw
function claimRefund(uint i, uint amount) external {
    require(
        amount > 0,
        "amount of token must be greater than zero"
    );
    require(

```

```

        campaigns[i].donation[msg.sender] >= amount,
        "You don't have enough to claim refund"
    );
    require(
        getStatus(i) == Status.during,
        "You only can claim refund when campaigns during"
    );

    campaigns[i].donation[msg.sender] -= amount;
    campaigns[i].collected -= amount;
    emit Refund(i, msg.sender, amount);
}

/// @notice Handle after campaign. Only allow campaign's owner run this function
/// @dev If campaign is succeed, campaign's owner will receive funds
/// @param i is index of campaign array
function endCampaign(uint i) external {
    require(
        msg.sender == campaigns[i].owner,
        "This function MUST be run by owner"
    );
    require(
        getStatus(i) == Status.succeed,
        "Campaign MUST be succeed"
    );
    require(
        campaigns[i].finstt == FinStatus.accepted,
        "Campaign MUST be accepted (NOT reject or paid)"
    );

    uint numStage = 1;
    uint amount = 0;
    bool isCompleted = false;
    (numStage, amount) = disb.getWithdrawInfo(
        i,
        campaigns[i].stage,
        campaigns[i].donors.length,
        campaigns[i].collected
    );
    if (numStage > 1) {
        if (amount > 0) {
            campaigns[i].stage += 1;
            if (campaigns[i].stage == numStage) {
                isCompleted = true;
            }
        } else {
            revert("Missing condition for withdraw");
        }
    } else {
        isCompleted = true;
        amount = campaigns[i].collected;
    }

    if (amount > 0) {
        if (isCompleted) {
            campaigns[i].finstt = FinStatus.paid;

```

```

        }
        emit Paid(i, msg.sender, amount);
        if(!token.addToken(msg.sender, amount)) {
            if (isCompleted) {
                campaigns[i].finstt = FinStatus.accepted;
            }
            if (numStage > 1) {
                campaigns[i].stage -= 1;
            }
        }
    }

/// @notice Get token donated for a campaign of donor
/// @param i is index of campaigns
/// @param donor is address of donor that you want to check
/// @return Number of token that donor donated for a campaign
function getDonation(uint i, address donor) external view returns(uint) {
    return campaigns[i].donation[donor];
}

/// @notice Get all amount of donor that invest to campaigns
/// @dev Get all amount of donor that donated and have checked campaign failed or
succeed
/// @param donor is address that you want check to amount of invest
/// @return Number of token that donated all campaigns
function getAllDonation(address donor) public view returns(uint) {
    uint tokens = 0;
    uint[] memory campaignsOf = donor2campaigns[donor];
    for (uint i = 0; i < campaignsOf.length; i++) {
        uint campID = campaignsOf[i];
        if (getStatus(campID) != Status.failed) {
            if (campaigns[campID].donation[donor] > 0) {
                tokens += campaigns[campID].donation[donor];
            }
        }
    }
    return tokens;
}

/// @notice Get list of campaign that donor donated
/// @param donor is address of donor
/// @return Array of campaign's id
function getCampaignList(address donor) external view returns(uint[] memory) {
    return donor2campaigns[donor];
}

/// @notice Get status of a campaign
/// @param i is index of campaigns array
/// @return {0 => during, 1 => failed, 2 => succeed, 3 => paid}
function getStatus(uint i) public view returns(Status) {
    if (now < campaigns[i].endDate) {
        return Status.during;
    } else {
        if (campaigns[i].collected < campaigns[i].goal) {

```

```
        return Status.failed;
    } else {
        return Status.succeeded;
    }
}

/// @notice Get financial status of campaign
/// @param i is index of campaign
/// @return {0 => pending, 1 => accepted, 2 => paid}
function getFinStatus(uint i) external view returns(FinStatus) {
    return campaigns[i].finstatus;
}

/// @notice Get number of campaigns
/// @return Number of campaigns
function length() external view returns(uint) {
    return campaigns.length;
}

}
```

Phụ lục C

Mã hợp đồng thông minh - Identity

Mã nguồn hợp đồng thông minh Identity được viết bằng ngôn ngữ Solidity:

```
pragma solidity ^0.5.3;

contract Identity {
    enum VerifyStatus {none, pending, verified, rejected}

    struct PersonalData {
        string name;
        string located;
        string privData;
        string shareKey;
        uint dob;
        VerifyStatus status;
    }
    struct VerifierData {
        string pubKey;
        uint task;
    }

    mapping (address => PersonalData) internal userInfo;
    mapping (address => bool) internal isVerifyRight;
    mapping (address => VerifierData) internal verifierInfo;
    mapping (address => address[]) internal verifier2users;
    address internal owner;
    address[] internal verifiers;

    /* -- Constructor -- */
    //
    /// @notice Constructor to create a Identity contract
    /// @dev Owner is runner of this contract
    constructor() public {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(
```

```

        msg.sender == owner,
        "Only owner"
    );
}

modifier onlyVerifier() {
    require(
        isVerifyRight[msg.sender] == true,
        "Only verifier");
}

/// @notice This function for user can register a identity
/// @param name is full name of user
/// @param located is located address of user
/// @param dob is date of birth of user
/// @param data is private data as hash of data that was store on IPFS
/// @param shareKey is secret key of user was encrypted
/// @param verifier is address of verifier
function registerIdentity(
    string calldata name,
    string calldata located,
    uint dob,
    string calldata data,
    string calldata shareKey,
    address verifier)
external {
    require(
        bytes(name).length > 3,
        "Your name is must be greater 3 characters"
    );

    require(
        bytes(located).length > 10,
        "Your located address must be greater 10 characters"
    );

    require(
        dob < now && dob > 0,
        "Date of birth is wrong"
    );

    require(
        userInfo[msg.sender].dob == 0,
        "You have already registered info"
    );

    require(
        isVerifyRight[verifier] == true,
        "Address verifier is incorrect");

    require(
        verifierInfo[verifier].task <= 10,
        "The verifier that you selected is no longer available"
    );
}

```

```

);

userInfo[msg.sender] = PersonalData(
    name,
    located,
    data,
    shareKey,
    dob,
    VerifyStatus.pending
);

verifier2users[verifier].push(msg.sender);
verifierInfo[verifier].task += 1;
}

/// @notice Get information of an user
/// @param user is address of user
/// @return Name, Located Address, Date of birth and verify status
function getIdentity(address user) external view
returns (
    string memory name,
    string memory located,
    uint dob,
    string memory privData,
    string memory shareKey,
    VerifyStatus status
) {
    name = userInfo[user].name;
    located = userInfo[user].located;
    dob = userInfo[user].dob;
    status = userInfo[user].status;
    privData = userInfo[user].privData;
    shareKey = userInfo[user].shareKey;
}
}

/// @notice This function for verifier to verify an identity
/// @param user is address of user
/// @param status is status include `true` is verified and `false` is rejected
function verify(address user, bool status) external onlyVerifier() {
    require(
        userInfo[user].status == VerifyStatus.pending,
        "User that you verify must be have data"
    );

    uint loopLimit = verifier2users[msg.sender].length;
    bool isVerifier2User = false;
    for (uint i = 0; i < loopLimit; i++) {
        if (verifier2users[msg.sender][i] == user) {
            isVerifier2User = true;
        }
    }
    require(
        isVerifier2User == true,
        "User must be requested verifier"
    );
}

```

```

);

userInfo[user].status = status ? VerifyStatus.verified : VerifyStatus.rejected;
verifierInfo[msg.sender].task -= 1;
}

/// @notice Get status of identity
/// @param user is address of user
/// @return Status (1 => pending, 2 => verified, 3 => rejected)
function getStatus(address user) external view returns(VerifyStatus) {
    return userInfo[user].status;
}

/// @notice This function for owner to add a verifier
/// @param verifier is address of verifier
/// @param pubKey is public key of verifier
function addVerifier(address verifier, string calldata pubKey) external onlyOwner() {
    require(
        isVerifyRight[verifier] == false,
        "This address have already added"
    );
    verifierInfo[verifier] = VerifierData(pubKey, 0);
    verifiers.push(verifier);
    isVerifyRight[verifier] = true;
}

/// @notice Get list all verifiers
/// @return array of verifier's addresses and count
function getVerifierAddresses() external view returns (address[] memory) {
    return verifiers;
}

/// @notice Get information of a verifier
/// @param verifier is address of verifier
/// @return Public key and number task of verifier
function getVerifier(address verifier) external view
returns(string memory pubKey, uint task) {
    pubKey = verifierInfo[verifier].pubKey;
    task = verifierInfo[verifier].task;
}

/// @notice Check identity of an address is verified
/// @param user is address of user
/// @return `true` if identity of address is verified
function isVerified(address user) external view returns(bool) {
    return userInfo[user].status == VerifyStatus.verified;
}

/// @notice Get list user that requested by Verifier
/// @return List of users
function getUsersRequested() external onlyVerifier() view returns(address[] memory) {
    return verifier2users[msg.sender];
}

```

```
/// @notice Function is used for other contract
/// @param verifier is address of user that you want to check
/// @return `true` if address is verifier
function isVerifier(address verifier) external view returns(bool) {
    return isVerifyRight[verifier];
}

/// @notice Change public key of verifier
/// @param verifier is address of verifier
/// @param newPubKey is new public key
function changePubKey (address verifier, string calldata newPubKey)
external onlyOwner {
    verifierInfo[verifier].pubKey = newPubKey;
}

/// @notice Function is used to check if owner
/// @return `true` if sender is owner of contract
function isOwner() external view returns(bool) {
    return msg.sender == owner;
}
}
```

Phụ lục D

Mã hợp đồng thông minh - Disbursement

Mã nguồn hợp đồng thông minh Disbursement được viết bằng ngôn ngữ Solidity:

```
pragma solidity ^0.5.3;
import {Campaigns} from './Campaigns.sol';

contract Disbursement {
    enum Vote {none, Agree, Disagree}
    enum Mode {Flexible, Fixed, TimingFlexible, TimingFixed}
    struct Data {
        uint numStage;
        uint[] amount;
        uint[] time;
        uint[] agreed;
        uint[] amountAgreed;
        Mode mode;
        mapping(address => mapping(uint => Vote)) voting; // user => stage => Vote
    }

    Campaigns internal camp;
    mapping (uint => Data) internal stages;

    /* -- Constructor -- */
    //
    /// @notice Constructor run only one time
    /// @dev This contract MUST be run after TokenSystem
    /// @param addrCampaign is address of Campaigns contract
    constructor(Campaigns addrCampaign) public {
        camp = addrCampaign;
    }

    /// @notice only some contracts MUST be run
    modifier onlyAllowedContract() {
        require(
            msg.sender == address(camp),
            "Only allow linked contract"
        );
    }
}
```

```

}

/// @notice Create disbursement for campaign
/// @dev This function must be run by Campaigns contract
/// @param campID is campaign's id
/// @param numStage is number of stage
/// @param amount is array amount for each stages (unit: tokens, sum all must be
equal with campaign's goal)
/// @param mode is MODE for disbursement (type and requirement for withdraw)
/// @param time is array of time for each stages (unit: seconds, start from campaign
's end date). Notice: first element is default with zero
function create(
    uint campID,
    uint numStage,
    uint[] calldata amount,
    Mode mode,
    uint[] calldata time
)
external onlyAllowedContract() {
    stages[campID] = Data(
        numStage,
        amount,
        time,
        new uint[](numStage),
        new uint[](numStage),
        mode
    );
}

/// @notice Return disbursement info of a campaign
/// @param campID is campaign's id
/// @return Some info as number of stage, array of amount, mode, array of time,
array of number agree voted
function getInfo(uint campID) external view
returns (
    uint numStage,
    uint[] memory amount,
    Mode mode,
    uint[] memory time,
    uint[] memory agreed) {
    numStage = stages[campID].numStage;
    amount = stages[campID].amount;
    mode = stages[campID].mode;
    time = stages[campID].time;
    agreed = stages[campID].agreed;
}

/// @notice This function for backer to vote for a stage of campaign disbursement
/// @param campID is campaign's id
/// @param stage is stage number (start with 1. Stage 0 default withdraw without
voting)
/// @param isAgree is decision for vote (Two options: `true` for agree withdraw,
otherwise for disagree)
function vote(uint campID, uint stage, bool isAgree) external {
    require(

```

```

        stage > 0,
        "Stage 0 is default full withdraw without voting"
    );
    require(
        stage < stages[campID].numStage,
        "Stage value is invalid"
    );
    require(
        stages[campID].voting[msg.sender][stage] == Vote.none,
        "You already voted for this stage"
    );
    require(
        camp.getDonation(campID, msg.sender) > 0,
        "You don't have right for this action"
    );

    require(
        !(stages[campID].mode >= Mode.TimingFlexible &&
        now < stages[campID].time[stage]),
        "Don't have enough time to do this action"
    );

    if (isAgree == true) {
        stages[campID].voting[msg.sender][stage] = Vote.Agree;
        stages[campID].agreed[stage] += 1;
        stages[campID].amountAgreed[stage] += camp.getDonation(campID, msg.sender);
    } else {
        stages[campID].voting[msg.sender][stage] = Vote.Disagree;
    }
}

/// @notice This function return info about withdraw campaign related with multi-
/// stage disbursement
/// @dev This function was run by Campaigns contract
/// @param campID is campaign's id
/// @param stage is stage number (start with 0)
/// @param numberofDonors is number of backers (person that backed to campaign)
/// @param campCollected is total token that campaign was collected
/// @return Two values: (1) number of stage; (2) amount for withdraw, if don't meet
/// condition, amount will have value is zero
function getWithdrawInfo(uint campID, uint stage, uint numberofDonors, uint
campCollected)
external view returns (
    uint numStage,
    uint amount
) {
    if (stages[campID].numStage > 0) {
        numStage = stages[campID].numStage;
        if (stage > 0) {
            uint agreed = stages[campID].agreed[stage];
            uint amountAgreed = stages[campID].amountAgreed[stage];
            uint minAgree = numberofDonors / 2;
            uint minAmount = campCollected / 2;
            amount = agreed > minAgree && amountAgreed > minAmount ? stages[campID].
amount[stage] : 0;
        }
    }
}

```

```

    if (
        (stages[campID].mode == Mode.Fixed ||
        stages[campID].mode == Mode.TimingFixed) &&
        stage > 1 &&
        stages[campID].agreed[stage-1] <= minAgree &&
        stages[campID].amountAgreed[stage-1] <= minAmount
    ) {
        amount = 0;
    }
    if (
        stages[campID].mode >= Mode.TimingFlexible &&
        now < stages[campID].time[stage]
    ) {
        amount = 0;
    }
} else { // stage 0 is default full withdraw without voting
    amount = stages[campID].amount[0];
}
} else {
    numStage = 1;
    amount = 0;
}

}
}

```