Hướng tới cơ sở hạ tầng khóa công khai, phi tập trung, dựa trên hợp đồng thông minh[1]

Christos Patsonakis[*], Katerina Samari[2*], Mema Roussopoulos[*] và Aggelos Kiayias[2†]

[*]Đại học Quốc gia và Kapodistrian Athens, Hy Lạp ,
{c.patswnakis,ksamari,mema}@di.uoa.gr

[†]Đại học Edinburgh và IOHK, Vương quốc Anh,
Aggelos.Kiayias@ed.ac.uk

Tóm lược

Cơ sở hạ tầng khóa công khai (PKI) là một phần không thể thiếu trong nền tảng bảo mật của trao đổi kỹ thuật số. Việc triển khai rộng rãi PKI đã cho phép sự phát triển của các ứng dụng quan trọng, chẳng hạn như ngân hàng trực tuyến và thương mại điện tử. PKI tập trung (CPKI) dựa vào hệ thống phân tầng của Cơ quan chứng nhận (CA) đáng tin cậy để cấp, phân phối và quản lý trạng thái *chứng chỉ kỹ thuật số*, tức là các cấu trúc dữ liệu không thể giả mạo để chứng thực tính xác thực khóa công khai của thực thể. Thật không may, CPKI có nhiều nhược điểm về bảo mật và khả năng chịu lỗi, và đã xảy ra nhiều sự cố bảo mật trong suốt những năm qua. PKI phi tập trung (DPKI) đã được đề xuất để giải quyết những vấn đề này vì chúng dựa vào nhiều Node độc lập. Tuy nhiên, việc phi tập trung làm nảy sinh những mối lo ngại khác như đâu là động lực cho các Node tham gia để đảm bảo tính khả dụng của dịch vụ.

Chúng tôi tận dụng khả năng mở rộng cũng như cơ chế khuyến khích tích hợp của hệ thống Blockchain và đề xuất DPKI dựa trên hợp đồng thông minh. Rào cản chính trong việc hiện thực hóa DPKI dựa trên hợp đồng thông minh là quy mô trạng thái của hợp đồng, vốn là nguồn truy cập đắt nhất, cần được giảm thiểu để thiết kế có thể tồn tại được. Chúng tôi giải quyết vấn đề này bằng cách đề xuất và sử dụng trong DPKI của mình một bộ tích lũy mật mã *trạng thái công khai* với kích thước không đổi, một công cụ mật mã có thể được quan tâm độc lập trong bối cảnh các giao thức Blockchain. Chúng tôi cũng là những người đầu tiên chính thức hóa vấn đề thiết kế DPKI trong khuôn khổ Khả năng kết hợp phổ quát (UC) và chính thức chứng minh tính bảo mật cho thiết kế của chúng tôi theo giả định RSA mạnh mẽ trong mô hình Oracle ngẫu nhiên và sự tồn tại của hàm hợp đồng thông minh lý tưởng.

1 Giới thiệu

Mật mã khóa công khai hay bất đối xứng là một khối xây dựng quan trọng để bảo mật các thông tin liên lạc quan trọng trên Internet, chẳng hạn như thương mại điện tử và ngân hàng Internet. Để kích hoạt các ứng dụng như vậy, cơ sở hạ tầng khóa công khai (PKI) là cần thiết vì chúng cung cấp ánh xạ có thể xác minh được từ tên của thực thể đến khóa công khai tương ứng. Về bản chất, PKI là một hệ thống cho phép tạo, thu hồi, lưu trữ và phân phối *chứng chỉ kỹ thuật số*, tức là các cấu trúc dữ liệu không thể giả mạo để chứng thực tính xác thực khóa công khai của thực thể.

[1]Đây là phiên bản đầy đủ của nghiên cứu [42].

[2]Nghiên cứu này được hỗ trợ bởi dự án ERC CODAMODA và IOHK.

Trong PKI tập trung (CPKI), Cơ quan chứng nhận (CA), chịu trách nhiệm cấp, phân phối và quản lý trạng thái của chứng chỉ kỹ thuật số. Hai giả định phải được đưa ra khi triển khai CPKI. Đó là: 1) mọi người đều biết khóa công khai (chính xác) của CA và, 2) các câu lệnh được ký bởi khóa riêng tư của CA là hợp lệ, tức là mọi người đều tin cậy CA. Trong CPKI, việc đăng ký được xử lý theo hai giai đoạn. Trong giai đoạn đầu tiên, người dùng chứng minh yêu cầu của mình về danh tính với Cơ quan đăng ký (RA). Giả sử RA xác nhận yêu cầu, nó sẽ chuyển tiếp yêu cầu của người dùng tới CA. Trong giai đoạn thứ hai, người dùng nhận được chứng chỉ kỹ thuật số được ký bằng khóa riêng tư của CA, do đó chứng thực tính hợp lệ của nó. CA *định kỳ* xuất bản các cấu trúc dữ liệu đã ký có chứa chứng chỉ bị thu hồi, ví dụ: danh sách thu hồi chứng chỉ (CRL). Việc phân phối thông tin liên quan đến chứng chỉ được xử lý bởi CA (CA trực tuyến) hoặc được ủy quyền cho các danh mục trực tuyến, có thể truy cập công khai (CA ngoại tuyến).

Mặc dù được sử dụng chủ yếu nhưng CPKI vẫn có một số nhược điểm. CA tạo thành một điểm lỗi duy nhất, cả về tính bảo mật và tính khả dụng. Đã có một số sự cố CA bị tấn công dẫn đến việc cấp chứng chỉ giả cho tên miền của các tập đoàn nổi tiếng, chẳng hạn như Google ([3]). Các ví dụ nổi bật khác là sự cố Symantec ([4]) và TrustWave ([8]), cũng như mối lo ngại ngày càng tăng của chính phủ và các tổ chức tư nhân về việc có thể cấp chứng chỉ giả để giám sát, do đó vi phạm quyền riêng tư của người dùng cuối ([50]). Trong thực tế, tồn tại nhiều CA được liên kết với các mối quan hệ thân thiết được xác định rõ ràng, dựa trên sự tin cậy và các chính sách khác. Ví dụ đáng chú ý nhất của thiết kế này là chuỗi chứng chỉ SSL/TLS. Mô hình chứng nhận phân tầng, giống như hình cây được thiết kế để tăng khả năng mở rộng và khả năng chịu lỗi của hệ thống. Tuy nhiên, sự thoả hiệp của CA gốc hoặc thậm chí CA cấp dưới vẫn là một mối lo ngại ([26]).

Trong PKI phi tập trung (DPKI), nhiều Node độc lập hợp tác và cung cấp cùng một bộ dịch vụ mà không cần dựa vào một hoặc nhiều bên thứ ba đáng tin cậy (TTP). DPKI đã được đề xuất bởi vì với tư cách là hệ thống phân tán, chúng có tiềm năng cung cấp một số đặc tính mong muốn mà CPKI không thể cung cấp, ví dụ như khả năng mở rộng, khả năng chịu lỗi, cân bằng tải và tính sẵn sàng. Các nhà nghiên cứu đã đề xuất DPKI dựa trên nhiều nguyên thủy phân tán khác nhau, chẳng hạn như bảng Hash phân tán (DHT) (ví dụ: [9]). Để phát hiện các Node độc hại và tăng cường bảo mật, chúng sử dụng các giao thức chia sẻ bí mật, ngưỡng và thỏa thuận Byzantine (ví dụ: [11, 23]). Những kỹ thuật này mặc dù phức tạp hơn để thiết kế và triển khai chính xác, nhưng lại tạo ra các hệ thống không có điểm lỗi duy nhất. Thật không may, các DPKI trước đây không cung cấp cơ chế khuyến khích cho các Node tham gia để đảm bảo dịch vụ được cung cấp vẫn sẵn sàng trong thời gian dài, ví dụ: chúng không giải quyết được vấn đề *Free-riding* ([33]), một vấn đề tồn tại khi người dùng tận dụng các dịch vụ hoặc lợi ích từ mạng Blockchain mà không đóng góp đủ công sức hoặc tài nguyên của chính họ.

Các giao thức Blockchain (ví dụ: Bitcoin [39]), có cơ chế khen thưởng khuyến khích các bên tham gia vào giao thức. Phần thưởng đến dưới dạng một loại tiền kỹ thuật số để đền bù cho những người tham gia, do đó, tạo ra động cơ phản đối việc Free-riding, trong khi vẫn duy trì một hệ thống tham gia miễn phí, có khả năng mở rộng cao.

Trong nghiên cứu này, chúng tôi trình bày thiết kế DPKI trên nền tảng hợp đồng thông minh, một thế hệ Blockchain mới cho phép phát triển hợp đồng thông minh, tức là các tác nhân có trạng thái "đang hoạt động" trong Blockchain có thể thực thi các hàm chuyển đổi trạng thái tùy ý. Rào cản chính trong việc hiện thực hóa DPKI dựa trên hợp đồng thông minh là quy mô trạng thái của hợp đồng thông minh, vốn là nguồn truy cập đắt nhất, cần được giảm thiểu để việc xây dựng được coi là khả thi. Các giải pháp dựa trên Blockchain trước đây, chẳng hạn như Namecoin ([6]) và Emercoin ([2]), thiếu phần này vì trạng thái của chúng là tuyến tính với số lượng thực thể đã đăng ký. Fromknecht và

cộng sự [27] cải thiện điều này bằng cách khai thác sức mạnh của bộ tích lũy mật mã, tức là cấu trúc dữ liệu tiết kiệm không gian cho phép truy vấn tư cách thành viên. Tuy nhiên, chúng tôi tin rằng họ không khai thác hết tiềm năng vì những lý do sau: 1) trạng thái hệ thống của họ vẫn có độ phức tạp Logarit, do sử dụng bộ tích lũy dựa trên cây Merkle và, 2) cấu trúc của họ tính toán lại các giá trị tích lũy để xử lý việc xóa các phần tử, tức là mỗi lần xóa (thu hồi) có độ phức tạp tính toán tuyến tính. Chúng tôi giải quyết những sự thiếu hiệu quả này bằng cách trình bày một cấu trúc có trạng thái không đổi và tránh tính toán lại các giá trị tích lũy. Khối xây dựng chính của chúng tôi là bộ tích lũy phổ quát, trạng thái công khai, dựa trên giả định RSA mạnh mẽ trong mô hình Oracle ngẫu nhiên, trong số những mô hình khác, nó có các đặc tính tốt sau: 1) bộ tích lũy và các cấu trúc để chứng minh tư cách thành viên và không phải thành viên (được gọi là *bằng chứng*) có kích thước không đổi và, 2) tất cả các hoạt động của nó có thể được thực hiện một cách hiệu quả bằng cách chỉ có quyền truy cập vào khóa công khai của bộ tích lũy.

Tóm lại, những đóng góp của nghiên cứu này như sau:

- Chúng tôi đề xuất thiết kế DPKI trên nền tảng hợp đồng thông minh. Do khả năng tương tác của các hợp đồng thông minh, hệ thống của chúng tôi cung cấp một cơ chế chung để xác thực trên Blockchain, cho đến thời điểm này, được xử lý theo cách đặc biệt. Hơn nữa, tính chất có thể lập trình của các nền tảng này cho phép chúng tôi phát triển hệ thống với các nguyên thủy hiệu quả hơn, khi chúng có sẵn mà không cần phân nhánh trong Blockchain, đó là trường hợp của các Blockchain PKI chuyên dụng.

- Chúng tôi giải quyết rào cản chính trong việc hiện thực hóa DPKI dựa trên hợp đồng thông minh khả thi bằng cách cung cấp một cấu trúc có thuộc tính “không đổi”, tức là cả trạng thái của hợp đồng thông minh cũng như các cấu trúc để chứng minh tư cách thành viên và không phải thành viên đều có kích thước không đổi. Chúng tôi nhấn mạnh tầm quan trọng của đặc tính này vì nó đảm bảo tính hiệu quả và đảm bảo chi phí tiền tệ kỹ thuật số thống nhất cho mọi hoạt động cụ thể nào đối với tất cả người dùng, tức là sự công bằng về mặt chi phí.

- Thiết kế của chúng tôi dựa trên một bộ tích lũy trạng thái công khai, bổ sung, phổ quát, một công cụ mã hóa có thể được quan tâm độc lập đối với các giao thức sử dụng Blockchain để xác minh tính hợp lệ của thông tin một cách hiệu quả.

- Chúng tôi là những người đầu tiên chính thức hóa vấn đề thiết kế DPKI trong khuôn khổ Khả năng kết hợp toàn cầu (UC) ([19]). Chúng tôi chính thức chứng minh tính bảo mật cho công trình của mình theo giả định RSA mạnh mẽ trong mô hình Oracle ngẫu nhiên và sự tồn tại của một hàm hợp đồng thông minh lý tưởng.

- Mặc dù ứng dụng dự kiến của chúng tôi là PKI, nhưng chúng tôi mô hình cụ thể dịch vụ của mình dưới dạng “Dịch vụ đặt tên” chung. Do đó, thiết kế của chúng tôi có thể được chuyển sang triển khai một cách hiệu quả các dịch vụ khác nằm trong mô hình này, ví dụ: hệ thống tên miền phân tán (DDNS).

2 Công việc liên quan

Một số hệ thống được đề xuất trước đây sử dụng cùng một nguyên tắc cơ bản, mỗi hệ thống có cách riêng để phi tập trung các dịch vụ của PKI. Vì lợi ích về không gian, chúng tôi tập trung vào các DPKI chính thức, tức là các hệ thống thực hiện đăng ký, thu hồi, lưu trữ và truy xuất chứng chỉ. Do đó, chúng tôi sẽ không quan tâm đến các hệ thống chứng nhận (ví dụ: [35]), không cung cấp phương pháp thu hồi, kết hợp, ví dụ: ghép CA với lớp phủ có cấu trúc (ví dụ: [48]), hoặc thậm chí PGP ([52]), hoạt động dựa trên các máy chủ tập trung. Chúng tôi cũng xem xét công việc liên quan đến bộ tích lũy mật mã, vốn là cơ sở cho thiết kế của chúng tôi.

Các nhà nghiên cứu đã đề xuất DPKI dựa trên mô hình máy trạng thái nhân rộng (RSM) ([51, 43]) để thực thi quan điểm nhất quán, toàn cầu về trạng thái của hệ thống. Điều này đạt được bằng cách để các Node tham gia vào một giao thức thỏa thuận đã được xác thực và thường giả định: 1) ngưỡng *t* của các Node bị lỗi, 2) giao thức *join()* và *leave()* cho các Node muốn tham gia hoặc rời khỏi nhóm mô phỏng để điều chỉnh tham số ngưỡng của hệ thống và 3) các Node có thể xác thực bất kỳ người tham gia (tiềm năng) nào. Trong PKI dựa trên RSM, việc đăng ký yêu cầu một bên tham gia phải thực hiện đàm phán "ngoài kênh chính" với nhiều miền quản trị, điều này gây khó khăn cho người dùng. Ngoài ra, tính không xác định, ví dụ như ghi dấu thời gian, là một khó khăn chính của việc mô phỏng nhất quán vì nó có thể dẫn đến sự phân kỳ trạng thái mô phỏng, do đó làm ảnh hưởng đến khả năng chịu lỗi. Tuy nhiên, việc dán nhãn thời gian là cần thiết trong PKI để theo dõi thời gian tồn tại của chứng chỉ. Mặt khác, các hệ thống dựa trên Blockchain không gặp phải vấn đề này và chúng đã được sử dụng để triển khai các dịch vụ ghi dấu thời gian (ví dụ: [31]). Hơn nữa, họ sử dụng một hình thức thỏa thuận khác dựa trên tính toán. Thuật toán thỏa thuận thay thế này có đặc tính tốt là có thể thích ứng khi các Node tự do tham gia và rời khỏi hệ thống. Kinh nghiệm đã minh họa rằng cách tiếp cận Blockchain đã được cả cộng đồng nghiên cứu cũng như ngành công nghiệp đánh giá cao do tính chất có khả năng mở rộng, thích ứng và không hạn chế cao ([1, 5, 7]).

Lớp phủ có cấu trúc cũng đã được đề xuất để phân phối các dịch vụ của PKI ([11, 23]). Theo thiết kế, chúng có khả năng mở rộng, cân bằng tải, cung cấp khả năng lưu trữ và truy xuất dữ liệu hiệu quả. Thật không may, những hệ thống này không chống lại được các cuộc tấn công của Sybil. Douceur ([25]) đã chứng minh rằng để chống lại cuộc tấn công Sybil, các hệ thống phân tán phải sử dụng khả năng xác thực hoặc sức mạnh tính toán. Tuy nhiên, các hệ thống dựa trên DHT nói trên cũng không sử dụng, do đó chúng không an toàn. Ngược lại, các Blockchain có khả năng phục hồi trước cuộc tấn công Sybil vì hoạt động của chúng vốn phụ thuộc vào sức mạnh tính toán.

Trong tất cả các hệ thống trên, các Node dự kiến sẽ tham gia vào các giao thức sử dụng nhiều tài nguyên. Thật không may, những hệ thống này không khuyến khích sự tham gia của Node cũng như không thực thi hành vi đúng đắn của các Node tham gia.

Cách tiếp cận ban đầu của thiết kế PKI dựa trên Blockchain dựa vào việc có sự tương đồng cố hữu giữa các dịch vụ của DNS và PKI tương ứng. Về cơ bản, cả hai đều ánh xạ tên định danh tới một giá trị nào đó (có thể là địa chỉ IP hoặc khóa công khai). Một trong những Altcoin *lớn nhất*, Namecoin ([6]), cung cấp DNS phân tán làm hàm chính. Trong Namecoin, Blockchain được sử dụng để lưu trữ cũng như xác minh/truy vấn các bản ghi DNS. Một số DPKI tuân theo cách tiếp cận này, ví dụ đáng chú ý nhất là Emercoin ([2]). Thật không may, cách tiếp cận này không hiệu quả vì nó buộc mỗi người dùng phải lưu trữ toàn bộ bản sao của Blockchain và duyệt qua nội dung của nó mỗi khi cần xác thực ánh xạ. Điều này hạn chế đáng kể khả năng ứng dụng của hệ thống; ví dụ: việc lưu trữ toàn bộ

Blockchain trên điện thoại thông minh là không thể. Hơn nữa, việc xác thực ánh xạ, hoạt động thường xuyên nhất, đòi hỏi lượng tính toán ngày càng tăng khi có nhiều Block được thêm vào Blockchain.

Một cách tiếp cận hiện đại, liên quan hơn là sử dụng bộ tích lũy mật mã, lần đầu tiên được giới thiệu trong nghiên cứu của Benaloh và cộng sự [15] như một giải pháp thay thế phi tập trung cho chữ ký số. Đây là những cấu trúc dữ liệu tiết kiệm không gian cho phép truy vấn thành viên. Cấu trúc ban đầu của chúng đã được cải tiến trong nghiên cứu của Bari và cộng sự [14] bằng cách củng cố khái niệm bảo mật ban đầu về thuộc tính *không xảy ra va chạm*. Camenisch và cộng sự [18] đã mở rộng các tài liệu trước đó và trình bày sơ đồ tích lũy đầu tiên cho phép các phần tử được thêm/xóa động, dựa trên giả định RSA mạnh mẽ. Trong sơ đồ này, các bằng chứng thành viên có thể được cập nhật bằng cách chỉ sử dụng khóa công khai của bộ tích lũy, nghĩa là không cần thông tin bí mật. Sau nghiên cứu này, nhiều cách xây dựng khác nhau đã được đề xuất trong tài liệu với những đặc điểm khác nhau và dựa trên những giả định khác nhau. Li và cộng sự [36] đã giới thiệu khái niệm về *bộ tích lũy phổ quát*, tức là bộ tích lũy hỗ trợ cả truy vấn thành viên và không phải thành viên, và đề xuất cấu trúc dựa trên RSA cho bộ tích lũy phổ quát. Các sơ đồ tích lũy được đề xuất khác dựa trên cây Merkle (ví dụ: [41], [16], [44]), các cặp song tuyến tính (ví dụ: [40], [10], [17], [22]) và lưới (ví dụ: [32], [49]). Một phân loại chi tiết về các sơ đồ tích lũy hiện có có thể được tìm thấy trong [24], trong đó đề xuất một mô hình chính thức thống nhất cho các bộ tích lũy mật mã.

Bộ tích lũy mật mã cung cấp một số lợi ích. Đầu tiên, kích thước nhỏ gọn (hoặc thậm chí không đổi) khiến chúng trở thành ứng viên phù hợp cho các thiết bị có dung lượng lưu trữ hạn chế (ví dụ: điện thoại thông minh). Thứ hai, hầu hết các xác minh tư cách thành viên và không phải thành viên đều có chi phí tính toán không đổi, bất kể số lượng giá trị tích lũy (tích lũy là việc thêm một phần tử vào bộ tích lũy). Thứ ba, các đặc tính bảo mật của chúng dựa trên các giả định về độ cứng tiêu chuẩn, do đó, khiến chúng phù hợp với các cơ sở hạ tầng bảo mật quan trọng. Một lợi ích nữa của việc xây dựng dựa trên bộ tích lũy là chúng không sử dụng Blockchain để thực thi sự đồng thuận trên toàn bộ tập hợp ánh xạ (danh tính, khóa công khai), như trường hợp của Namecoin và Emercoin. Thay vào đó, đối tượng đồng thuận là (các) giá trị tích lũy, có các lợi ích sau. Đầu tiên, người dùng không bắt buộc phải thực hiện truy xuất và xác minh đầy đủ toàn bộ lịch sử giao dịch, tức là tải xuống và xác thực toàn bộ Blockchain. Thay vào đó, người dùng cũ hoặc người dùng mới chỉ có thể tải xuống và xác thực các tiêu đề khối để cập nhật trạng thái, điều này hiệu quả hơn nhiều cả về mặt giao tiếp và tính toán. Thứ hai, nó cho phép giới thiệu một thành phần không cần phải tin cậy mà người dùng có thể truy vấn để có được một phiên bản nhỏ gọn hơn của toàn bộ lịch sử hoạt động một cách hiệu quả, so với toàn bộ lịch sử giao dịch. Do tính chất có thể kiểm chứng của bộ tích lũy mật mã, hiệu quả tăng lên này không mất phí.

Certcoin ([27]) là một PKI dựa trên Blockchain nhằm giải quyết sự kém hiệu quả đã nói ở trên của Namecoin và Emercoin bằng cách tách việc lưu trữ thông tin khỏi xác minh của nó. Nó sử dụng DHT được xác thực để lưu trữ chứng chỉ kỹ thuật số, dựa trên Kademlia ([38]). Các mạng này tạo điều kiện thuận lợi cho việc lưu trữ và truy xuất các truy vấn có độ phức tạp Logarit, nghĩa là chúng rất hiệu quả. Hơn nữa, bản chất xác thực giúp nó an toàn trước cuộc tấn công Sybil ([25]). Để tạo điều kiện thuận lợi cho việc xác minh ánh xạ (danh tính, khóa công khai), Certcoin duy trì hai bộ tích lũy mật mã trong Blockchain. Công cụ tích lũy đầu tiên của Certcoin dựa trên giả định RSA mạnh mẽ và tích lũy tên định danh. Do đó, người dùng có thể kiểm ta xem danh tính đã được đăng ký trong hệ thống hay chưa. Bộ tích lũy thứ hai dựa trên cây Merkle (được trình bày chính thức trong [44]) và tích lũy ánh xạ (danh tính, khóa công khai). Điều này cho phép máy trạm xác thực tính xác thực của

bất kỳ ánh xạ nào được lấy từ mạng DHT bằng cách tải xuống và xác thực Block mới nhất cũng như bằng cách thực hiện các truy vấn thành viên thích hợp.

Sau đây, chúng tôi nêu bật những điểm khác biệt trong thiết kế của chúng tôi so với Certcoin. Chúng tôi mở đầu cuộc thảo luận với các vấn đề liên quan đến bộ tích lũy. Đầu tiên, thiết kế của chúng tôi sử dụng hai bộ tích lũy dựa trên RSA, có kích thước không đổi và khóa công khai nhỏ. Ngược lại, Certcoin sử dụng một RSA và một bộ tích lũy dựa trên cây Merkle. Kích thước của bộ tích lũy dựa trên cây Merkle tăng lên khi có nhiều phần tử được tích lũy hơn, tức là nó không cố định. Đây là một vấn đề trong lĩnh vực Blockchain vì các thợ đào thích các Block nhỏ có thể được Hash nhanh hơn và giảm chi phí vận hành để tăng lợi nhuận. Do đó, chúng ta sẽ rất khó để khuyến khích các thợ đào hỗ trợ Blockchain của Certcoin có các Block có kích thước thay đổi. Hơn nữa, trong các hệ thống dựa trên Blockchain, chi phí thực hiện giao dịch phụ thuộc vào quy mô của chúng. Vì vậy, Certcoin không đảm bảo *sự công bằng* về mặt chi phí giao dịch. Thiết kế của chúng tôi không gặp phải những vấn đề này bởi vì nó có trạng thái không đổi. Thứ hai, trong khi các bộ tích lũy dựa trên cây Merkle có đặc tính tuyệt vời là các phần tử có thể bị xóa mà không cần biết về thông tin bí mật thì điều tương tự lại không có đối với các bộ tích lũy dựa trên RSA. Do đó, khi ánh xạ (danh tính, khóa công khai) bị thu hồi làm cho danh tính có sẵn trở lại, Certcoin sẽ tính toán lại bộ tích lũy RSA của nó từ đầu, điều này không hiệu quả. Để giải quyết thực tế là việc xóa trong bộ tích lũy RSA yêu cầu quyền truy cập vào khóa riêng tư, nếu được biết công khai, có thể phá vỡ tính bảo mật của chúng, chúng tôi sử dụng một thủ thuật được trình bày trong nghiên cứu của Baldimtsi và cộng sự [13]. Về cơ bản, chúng tôi sử dụng thẻ để đánh dấu các thành phần là "đã thêm" (trong khi đăng ký) hoặc "đã xóa" (trong khi thu hồi). Do đó, trái ngược với Certcoin, chúng tôi không tính toán lại bất kỳ bộ tích lũy nào. Thứ ba, Certcoin kết hợp chặt chẽ quy trình khai thác với ứng dụng thực tế của Blockchain, nhằm cung cấp các dịch vụ của DPKI. Hai vấn đề này trực giao với nhau về mặt kiến trúc của hệ thống và chúng tôi tin rằng cần được giải quyết ở các lớp khác nhau. Thay vào đó, chúng tôi là người đầu tiên đề xuất DPKI dựa trên nền tảng hợp đồng thông minh, tức là các Blockchain có thể lập trình giúp tách giao thức đồng thuận của Blockchain khỏi các hàm của ứng dụng chạy trên nó. Sự khác biệt chính này cho phép chúng tôi phát triển hệ thống của mình với các nguyên thủy hiệu quả hơn, khi chúng có sẵn mà không cần Hardfork trong Blockchain, điều này không xảy ra đối với các Blockchain dành riêng cho ứng dụng, chẳng hạn như Certcoin. Ngoài ra, trong các nền tảng này, các hợp đồng thông minh có thể tương tác với nhau, do đó tạo ra một hệ sinh thái các ứng dụng có thể tương tác với nhau. Bằng cách tận dụng tính năng này, hệ thống của chúng tôi có thể cung cấp một cơ chế chung để xác thực trên Blockchain mà cho đến thời điểm này đã được xử lý theo cách đặc biệt. Thứ tư, Certcoin không có mô hình bảo mật cho PKI mà nó triển khai cũng như không có bằng chứng cho thấy nó cung cấp dịch vụ được yêu cầu. Ngược lại, chúng tôi chính thức hóa vấn đề thiết kế DPKI trong khung UC ([19]) và chúng tôi chính thức chứng minh tính bảo mật của nghiên cứu theo giả định RSA mạnh mẽ trong mô hình Oracle ngẫu nhiên. Canetti [20], cung cấp một công thức tối thiểu về chức năng của cơ quan cấp chứng chỉ lý tưởng hỗ trợ đăng ký và truy xuất khóa công khai. Công thức của chúng tôi có liên quan nhiều hơn và cho phép thực hiện nhiều thao tác hơn, chẳng hạn như thu hồi khóa công khai.

3 Sơ bộ

Ký hiệu. Chúng tôi sử dụng λ để biểu thị tham số bảo mật và **negl**$(\cdot)$ để biểu thị một hàm không đáng kể trong một vài tham số.

Định nghĩa 3.1 (Giả định RSA mạnh mẽ [14])**.** *Đối với bất kỳ kẻ tấn công ppt $\mathcal{A}$ nào,*

$$\mathbf{Pr}[n \leftarrow \mathbf{KeyGen}(1^{\lambda}); x \leftarrow \mathbb{Z}_n^*; (y, e) \leftarrow \mathcal{A}(n, x): y^e = x \mod n] = \mathbf{negl}(\lambda),$$

trong đó, n = pq, p và q là các số nguyên tố an toàn.

Định nghĩa 3.2 (Họ hàm Hash phổ quát 2 [21])**.** *Cho U = { f | f : X → Y } là một họ hàm. Chúng ta nói rằng U là Họ hàm băm phổ quát nếu, với mọi $x_1, x_2 \in X$ với $x_1 \neq x_2$ và với mọi $y_1, y_2 \in Y$,* $\mathbf{Pr}_{f \in U}[f(x_1) = y_1 \wedge f(x_2) = y_2] = \left(\frac{1}{|Y|}\right)^2$.

Định nghĩa 3.3 (Trình tạo giả định ngẫu nhiên)**.** *Cho $G : \{0, 1\}^k \rightarrow \{0, 1\}^{p(k)}$ là thuật toán thời gian đa thức xác định và $p(\cdot)$ là đa thức trong tham số k. Chúng ta nói rằng G là một trình tạo giả định ngẫu nhiên nếu, với bất kỳ thuật toán ppt D nào,*

$$\left|\mathbf{Pr}[D(r) = 1] - \mathbf{Pr}\left[D(G(s)) = 1\right]\right| \leq \mathbf{negl}(k),$$

trong đó, r là một chuỗi được chọn ngẫu nhiên thống nhất từ $\{0, 1\}^{p(k)}$ và s là hạt giống được chọn ngẫu nhiên thống nhất từ $\{0 , 1\}^{k}$.

4 Bộ tích lũy trạng thái công khai

Trong phần này, chúng tôi trình bày khối xây dựng chính của dịch vụ đặt tên. Cụ thể, trong Phần 4.1, chúng tôi cung cấp định nghĩa về bộ tích lũy trạng thái công khai, bổ sung, phổ quát. Trong Phần 4.2, chúng tôi trình bày cấu trúc cho bộ tích lũy như vậy theo giả định RSA mạnh mẽ trong mô hình Oracle ngẫu nhiên.

4.1 Định nghĩa về bộ tích lũy trạng thái công khai, bổ sung, phổ quát

Ở mức độ cao, chúng tôi coi bộ tích lũy là *trạng thái công khai*, nếu một bộ tích lũy có thể thực hiện tất cả các hoạt động bằng cách chỉ có quyền truy cập vào khóa công khai của nó, tức là không cần có kiến thức về thông tin bí mật. Theo thuật ngữ được trình bày trong [13], bộ tích lũy là *bổ sung* nếu nó chỉ cho phép bổ sung các phần tử, và là *phổ quát* nếu nó cho phép cả bằng chứng là thành viên và không phải thành viên. Sau đây, chúng tôi trình bày định nghĩa về bộ tích lũy trạng thái công khai, bổ sung, phổ quát. Định nghĩa của chúng tôi sử dụng hai bên đáng tin cậy. Bên đầu tiên, *T*, chạy thuật toán tạo khóa (**KeyGen**(1^{λ})) và xuất bản khóa công khai của bộ tích lũy. Bên thứ hai, "trình quản lý bộ tích lũy" T_{acc}, chịu trách nhiệm bảo trì bộ tích lũy[3].

Định nghĩa 4.1 (Bộ tích luỹ trạng thái công khai, bổ sung, phổ quát)**.** Gọi *D* là miền xác định của các phần tử tích lũy và *X* là tập hợp tích lũy hiện tại. Một bộ tích lũy phổ quát, trạng thái công khai, bổ sung bao gồm các thuật toán sau:

[3]Lưu ý rằng khái niệm bộ tích lũy trạng thái công khai khác với khái niệm bộ tích lũy mạnh, như được định nghĩa trong [16]. Trong một bộ tích lũy mạnh, thuật toán **KeyGen**, tạo ra giá trị ban đầu của bộ tích lũy, có thể thực thi công khai và bất kỳ bên nào cũng có thể xác minh tính hợp lệ của đầu ra của nó. Ngược lại, trong bộ tích lũy trạng thái công khai, thuật toán **KeyGen** được vận hành bởi bên đáng tin cậy *T*.

- **KeyGen**: Khi nhập tham số bảo mật λ, nó tạo ra một cặp khóa (pk, sk) và xuất ra pk. Thuật toán này được vận hành bởi T.

- **InitAcc**: Với đầu vào pk và tập hợp tích lũy trống $X = \emptyset$, nó xuất ra giá trị tích lũy c_0. Thuật toán này được vận hành bởi T_{acc}.

- **Add**: Với đầu vào pk, một phần tử $x \in D$ được thêm vào và một giá trị tích lũy c, nó xuất ra (c', W), trong đó c' là giá trị mới của bộ tích lũy và W, là bằng chứng thành viên cho x.

- **MemWitGen**: Với đầu vào pk, X, c và $x \in X$, nó xuất ra một bằng chứng thành viên W cho x.

- **NonMemWitGen**: Với đầu vào pk, X, c, x, trong đó $x \in D$ và $x \notin X$, nó xuất ra một bằng chứng không phải là thành viên W cho x.

- **UpdMemWit**: Với đầu vào pk, x, y, W, trong đó W là bằng chứng thành viên cho x, nó xuất ra bằng chứng thành viên được cập nhật W' cho x. Thuật toán này được chạy sau (c', W_y) ← **Add**(pk, y, c), trong đó W_y là bằng chứng thành viên của y.

- **UpdNonMemWit**: Với đầu vào pk, x, y, W, trong đó $x, y \in D$, $x \neq y$ và W là bằng chứng không phải thành viên cho x, nó xuất ra bằng chứng không phải thành viên được cập nhật W' cho x. Thuật toán này được chạy sau (c', W_y) ← **Add**(pk, y, c).

- **VerifyMem**: Với đầu vào pk, $x \in D$, W và c, nó xuất ra 1 hoặc 0.

- **VerifyNonMem**: Với đầu vào pk, $x \in D$, W và c, nó xuất ra 1 hoặc 0.

Một cách không chính thức, bộ tích lũy là *đúng* nếu đối với bất kỳ bằng chứng tư cách thành viên nào được tạo ra một cách trung thực, thuật toán xác minh tư cách thành viên cho ra 1, và nếu đối với bất kỳ bằng chứng không phải thành viên nào được tạo ra một cách trung thực, thuật toán xác minh không phải là thành viên cho ra 1. Hơn nữa, chúng tôi coi bộ tích lũy phổ quát là *an toàn* nếu không kẻ tấn công ppt nào có thể tạo ra một bằng chứng không phải là thành viên hợp lệ cho một thành viên của tập hợp tích lũy, cũng như không có bằng chứng thành viên hợp lệ cho một phần tử không phải là thành viên của tập hợp tích lũy. Thuộc tính bảo mật của bộ tích lũy có thể được đáp ứng là thuộc tính *không xảy ra va chạm* hoặc *tính đúng đắn* trong tài liệu. Định nghĩa chính thức về bảo mật được đưa ra dưới đây (Định nghĩa 4.2), sử dụng trò chơi giữa người thách thức $\mathcal{C}$ và kẻ tấn công $\mathcal{A}$, như minh họa trong Hình 1. Để có định nghĩa chính thức về tính đúng đắn, chúng tôi giới thiệu cho độc giả quan tâm đến [37].

Định nghĩa 4.2. *Chúng ta nói bộ tích lũy là an toàn nếu đối với bất kỳ kẻ tấn công ppt $\mathcal{A}$ nào tương tác với người thách thức $\mathcal{C}$, như được minh họa trong trò chơi bảo mật ở Hình 1, nó cho rằng* $\mathbf{Pr}\left[\mathcal{G}_{\mathcal{A}}^{\mathrm{acc-sec}}(1^{\lambda})\right]$ = **negl**(λ).

4.2 Thiết kế

Trong Hình 2, chúng tôi trình bày thiết kế của bộ tích lũy phổ quát, trạng thái công khai, bổ sung. Chúng tôi mong muốn tích lũy các danh tính hoặc cặp (danh tính, khóa công khai), tức là các chuỗi tùy ý. Do đó, miền tích lũy là $D = \{0, 1\}^*$. Thiết kế này là sự kết hợp của bộ tích lũy phổ quát dựa trên RSA của Li và cộng sự [36], kèm theo quy trình **Map**, ánh xạ chuỗi tuỳ ý thành số nguyên tố.

$\mathcal{G}_{\mathcal{A}}^{\text{acc-sec}}$:Với đầu vào$1^{\lambda}$

- $\mathcal{C}$ chạy **KeyGen**(1^{λ}), tạo ra (*pk*, *sk*) và đưa *pk* cho kẻ tấn công $\mathcal{A}$.
- Đầu tiên $\mathcal{A}$ thực hiện truy vấn **InitAcc** tới $\mathcal{C}$. $\mathcal{C}$ khởi tạo tập hợp $X \leftarrow \emptyset$, chạy **InitAcc** và trả về giá trị c_0 cho $\mathcal{A}$.
- Khi $\mathcal{A}$ thực hiện truy vấn **Add** cho một phần tử *x*, $\mathcal{C}$ đặt $X \leftarrow X \cup \{x\}$ và tính toán $(c', W) \leftarrow$ **Add**(*pk*, *x*, *c*). Sau đó, $\mathcal{C}$ trả về cặp (c', *W*) cho $\mathcal{A}$.
- $\mathcal{A}$ xuất ra (x^*, W^*).

Trò chơi trả về 1 nếu có ít nhất một trong các điều kiện sau:

1. $x^* \notin X$ và **VerifyMem**(*pk*, x^*, W^*, *c*) = 1,
2. $x^* \in X$ và **VerifyNonMem** (*pk*, x^*, W^*, *c*) = 1.

Hình 1: Trò chơi bảo mật giữa kẻ tấn công $\mathcal{A}$ và người thách thức $\mathcal{C}$, trong đó $\mathcal{C}$ đóng vai trò của cả *T* và T_{acc}.

Cụ thể, đối với bất kỳ thuật toán nào chạy trên đầu vào một phần tử $z \in \{0, 1\}^*$, trước tiên, bên chạy thuật toán sẽ thực hiện quy trình **Map**, ánh xạ *z* thành một số nguyên tố, ví dụ: z_p, sau đó tiếp tục bằng cách chạy thuật toán tương tự như trong bộ tích lũy của Li và cộng sự [36] cho số nguyên tố z_p. Quy trình **Map** mà chúng tôi sử dụng là phiên bản sửa đổi của quy trình được đề xuất trong [30]. Vì vậy, trước tiên chúng tôi trình bày quy trình được đề xuất trong [30], sau đó, trên cơ sở đó, chúng tôi trình bày thuật toán sửa đổi **Map**, được sử dụng trong thiết kế của Hình 2. Cuối cùng, chúng tôi chứng minh bộ tích lũy của Hình 2 là an toàn theo Định nghĩa 4.2.

Ánh xạ các chuỗi tùy ý thành số nguyên tố [30]. Gennaro và cộng sự [30] mô tả một quy trình sử dụng họ hàm Hash phổ quát *U* của các hàm (Định nghĩa 3.2), ánh xạ các chuỗi 3*k* bit thành chuỗi *k* bit với thuộc tính bổ sung là, với bất kỳ $y \in \{0, 1\}^k$ và cho $f \in U$, người ta có thể lấy mẫu thống nhất một cách hiệu quả từ tập $\{x \in \{0, 1\}^{3k}: f(x) = y\}$. Với đầu vào $z \in \{0, 1\}^*$, trước tiên nó tính $h(z)$, trong đó $h : \{0, 1\}^* \rightarrow \{0, 1\}^k$ là hàm Hash chống va chạm. Sau đó, nó lấy mẫu nhiều lần từ tập $\{x \in \{0, 1\}^{3k}: f(x) = h(z)\}$ để tìm số nguyên tố $O(k^2)$ lần. Quy trình này có khả năng chống va chạm nếu *h* có khả năng chống va chạm và sẽ tạo ra một số nguyên tố có xác suất cao do Bổ đề sau.

Bổ đề 4.1 ([30])**.** *Giả sử U là một Họ hàm băm phổ quát từ* $\{0, 1\}^{3k}$ *đến* $\{0, 1\}^k$*. Khi đó, với tất cả ngoại trừ một phân số* $(1/2^k)$ *của các hàm* $f \in U$ *và với mọi* $y \in \{0, 1\}^k$*, một phần của ít nhất* $1/ck$ *phần tử trong tập hợp* $\{x \in \{0, 1\}^{3k}: f(x) = y\}$ *là các số nguyên tố, với hằng số c nhỏ.*

Do đó, một thuật toán lấy mẫu ck^2 lần từ tập $\{x \in \{0, 1\}^{3k}: f(x) = h(z)\}$ sẽ không tìm được số nguyên tố chỉ với xác suất không đáng kể. Để hoàn thiện, chúng tôi cung cấp chứng minh Bổ đề 4.1 trong Phụ lục B. Chứng minh được trình bày trong Phụ lục B tương tự như chứng minh trong [47].

Miền của bộ tích lũy là $D = \{0, 1\}^*$.

- **KeyGen**: Với đầu vào 1^λ, nó tạo ra một cặp số nguyên tố an toàn p, q có độ dài bằng nhau, sao cho $p = 2p' + 1$, $q = 2q' + 1$ và p', q' cũng là số nguyên tố. Nó tính $n = pq$ và chọn g ngẫu nhiên từ QR_n. Nó thiết lập $\ell = \lfloor \lambda/2 \rfloor - 2$ và chọn một quy trình xác định **Map** (như được mô tả trong đoạn trước), nhận đầu vào là một chuỗi tùy ý và xuất ra một số nguyên tố nhỏ hơn $2^{\lfloor \lambda/2 \rfloor - 2}$. Nó thiết lập $pk = (n, g, \textbf{Map})$, $sk = (p, q)$ và xuất ra pk.
- **InitAcc**: Với đầu vào pk, nó xuất ra $c_0 = g$.
- **Add**: Với đầu vào pk, $x \in \{0, 1\}^*$ và c_p, nó gọi **Map** trên đầu vào x và nhận một số nguyên tố x_p. Sau đó, nó tính $c' = c^x \bmod n$, thiết lập $W = c$ và xuất ra (c', W).
- **MemWitGen**: Với đầu vào pk, X và $x \in \{0, 1\}^*$, nó tính toán và xuất ra $W = g^{\prod_{x_i \in X \setminus (x)} \textbf{Map}(x_i)}$
- **NonMemWitGen**: Với đầu vào pk, X, c, $x \notin X$, nó gọi **Map** với đầu vào x và nhận số nguyên tố x_p. Sau đó, nó tính $u = \prod_{x_i \in X} \textbf{Map}(x_i))$. Vì $gcd\,(x_p, u) = 1$, nên nó chạy thuật toán Euclide mở rộng và tính $a, b \in \mathbb{Z}$, sao cho $au + bx_p = 1$. Bằng phép chia Euclide, a có thể được viết dưới dạng $a = a' + qx_p$, trong đó $0 \le a' < x_p$. Do đó, $a'u + (b + qu)x_p = 1$. Nó đặt $b' = b + qu$ và tính $d = g^{-b'}$. Cuối cùng, nó đưa ra một bằng chứng không phải là thành viên $W = (a', d) = (a \bmod x, g^{-b-qu})$.
- **UpdMemWit**: Với đầu vào pk, x, y, W, nó gọi **Map** với đầu vào x và nhận số nguyên tố x_p. Sau đó, nó tính toán và xuất ra $W' = W^{x_p} \bmod n$.
- **UpdNonMemWit**: Với đầu vào pk, $x \notin X$,$y \in X$, c và $W = (a,d)$, nó gọi **Map** với đầu vào x, y và nhận các số nguyên tố x_p, y_p tương ứng. Vì $y_p \neq x_p$, nó chạy thuật toán Euclide mở rộng và tính a_0, $r_0 \in \mathbb{Z}$, sao cho $a_0 y_p + r_0 x_p = 1$. Sau đó, nó nhân cả hai vế với a, tức là $aa_0 y_p + ar_0 x_p = a$ và tính $a' = a_0 a \bmod x_p$. Sau đó, nó tìm $r \in \mathbb{Z}$, sao cho $a' y_p = a + rx_p$, tính $d' = dc^r \bmod n$ và xuất ra $W' = (a', d')$.
- **VerifyMem**: Với đầu vào pk, x, W, c, nó gọi **Map** với đầu vào x và nhận số nguyên tố x_p. Sau đó, nó xuất ra 1, nếu $W^{x_p} = c \bmod n$, nếu không, nó xuất ra 0.
- **VerifyNonMem**: Với đầu vào pk, x, W, c, trong đó $W = (a, d)$, nó gọi **Map** với đầu vào x và nhận số nguyên tố x_p. Sau đó, nó xuất ra 1, nếu $c^a = d^{x_p} g \bmod n$, nếu không, nó xuất ra 0.

Hình 2: Cấu trúc của bộ tích lũy phổ quát, trạng thái công khai.

Một phiên bản sửa đổi của thuật toán trong [30]. Trong thiết kế (Hình 2), chúng tôi sử dụng một phiên bản xác định của quy trình **Map** đã nói ở trên mà chúng tôi đề xuất bên dưới. Cụ thể, chúng tôi sử dụng trình tạo giả định ngẫu nhiên $G : \{0, 1\}^k \to \{0, 1\}^{p(k)}$, trong đó $p(k)$ là đa thức trong k và hàm Hash có nhãn h: $\{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^k$, có khả năng chống va chạm và được mô hình hóa như một Oracle ngẫu nhiên.

Chúng tôi bắt đầu bằng cách chọn hai nhãn $\textbf{label}_0, \textbf{label}_1 \in \{0, 1\}^*$. Sau đó, **Map** với đầu vào $z \in \{0, 1\}^*$, trước tiên tính $h(\textbf{label}_0, z)$. Tiếp theo, nó tính $G(h(\textbf{label}_1, z))$. Sau đó, nó lấy mẫu các phần tử từ tập hợp $\{x \in \{0, 1\}^{3k}: f(x) = h(\textbf{label}_0, z)\}$ sử dụng tính ngẫu nhiên $G(h(\textbf{label}_1, z))$ và dừng khi tìm thấy số nguyên tố. Dễ dàng thấy rằng vì $\textbf{label}_0, \textbf{label}_1$ giống nhau trong bất kỳ lần chạy thuật toán nào nên thuật toán với đầu vào $z \in \{0, 1\}^*$ luôn cho ra cùng một giá trị. Trong Bổ đề 4.2 dưới đây, chúng

tôi chứng minh **Map** tìm thấy một số nguyên tố, ngoại trừ với xác suất không đáng kể, và trong Bổ đề 4.3, chúng tôi chứng minh **Map** có khả năng chống va chạm.

Bổ đề 4.2. *Cho* $z \in \{0,1\}^*$. *Thuật toán* **Map** *với đầu vào z, xuất ra một số nguyên tố, ngoại trừ với xác suất không đáng kể, giả sử G là một trình tạo giả định ngẫu nhiên và hàm Hash h là một Oracle ngẫu nhiên.*

Chứng minh. Giả sử **Map** với đầu vào $z \in \{0,1\}^*$, không tìm được số nguyên tố có xác suất không đáng kể α. Chúng tôi sẽ xây dựng một ppt D riêng biệt, nó phá vỡ thuộc tính của trình tạo giả định ngẫu nhiên G như được định nghĩa trong Định nghĩa 3.3. Hãy nhớ lại rằng sự khác biệt duy nhất của quy trình trong [30] và thuật toán **Map** được mô tả trong đoạn trước là việc lấy mẫu các phần tử từ tập $X = \{x \in \{0,1\}^{3k} : f(x) = h(\mathbf{label}_0, z)\}$, tức là trong trường hợp trước, các phần tử được lấy mẫu ngẫu nhiên một cách thống nhất trong khi ở trường hợp sau, các phần tử được lấy mẫu bằng cách sử dụng ngẫu nhiên đầu ra của PRG G. Dựa vào đó, ta xét ppt D sau:

- Với đầu vào là một chuỗi $x \in \{0,1\}^{p(k)}$, lấy mẫu từ tập $X = \{x \in \{0,1\}^{3k} : f(x) = h(\mathbf{label}_0, z)\}$ sử dụng ngẫu nhiên trong chuỗi x.
- Nếu xuất ra số nguyên tố p thì trả về 1, nếu không thì trả về 0.

Đầu tiên, vì h là một Oracle ngẫu nhiên nên hạt giống $h(\mathbf{label}_1, z)$ được coi là ngẫu nhiên. Khi đó, nếu $x = r$, trong đó r được chọn thống nhất một cách ngẫu nhiên, theo Bổ đề 4.1, thì chúng ta có $\Pr[D(r) = 1] = 1 - \mathbf{negl}(k)$. Theo giả định **Map** không tìm được số nguyên tố có xác suất không đáng kể α, chúng ta có

$$\left|\mathbf{Pr}[D(r') = 1] - \mathbf{Pr}\left[G\left(h(\mathbf{label}_1, z)\right) = 1\right]\right| = 1 - \mathbf{negl}(k) - (1 - \alpha) = \alpha - \mathbf{negl}(k), \qquad (1)$$

Điều này mâu thuẫn với Định nghĩa 3.3.

□

Bổ đề 4.3. *Thuật toán* **Map** *có khả năng chống va chạm nếu hàm Hash h có khả năng chống va chạm. Cụ thể, không kẻ tấn công ppt nào có thể tìm thấy* $z_1, z_2 \in \{0,1\}^*$ *với* $z_1 \neq z_2$, *sao cho thuật toán* **Map** *trả về cùng một số nguyên tố p.*

Chứng minh. Giả sử **Map** không có khả năng chống va chạm, tức là có một kẻ tấn công ppt $\mathcal{A}$, tìm thấy hai z_1, z_2 khác nhau, sao cho $\mathbf{Map}(z_1) = \mathbf{Map}(z_2) = p$. Điều này đòi hỏi thuật toán **Map** lấy mẫu các phần tử từ cùng một tập nghiệm $X = \{x \in \{0,1\}^{3k} : f(x) = h(\mathbf{label}_0, z_1)\}$. Do đó, $\mathcal{A}$ sẽ tìm sự xung đột trong hàm Hash h, tức là $\mathcal{A}$ tìm z_1, z_2 sao cho $h(\mathbf{label}_0, z_1) = h(\mathbf{label}_0, z_2)$. Tuy nhiên, điều này chỉ đúng với xác suất không đáng kể. □

Tính bảo mật của bộ tích lũy trong Hình 2. Trước khi chính thức chứng minh tính bảo mật của bộ tích lũy trong Hình 2, chúng tôi đưa ra một ví dụ đơn giản về lý do tại sao quy trình **Map** phải mang tính tất định trong thiết kế của chúng tôi, giải thích theo hướng tại sao chúng tôi không thể sử dụng quy trình của [30] như hiện tại. Đầu tiên, giả sử chúng tôi đã sử dụng quy trình [30] mà không sửa đổi như đề xuất và phần tử $x \in \{0,1\}^*$ đã được thêm vào bộ tích lũy. Điều này có nghĩa là T_{acc} trước tiên tạo ra số nguyên tố x_p và sau đó thêm x_p vào bộ tích lũy RSA cơ bản. Sau đó, kẻ tấn công có thể tạo ra một bằng chứng không phải là thành viên W cho x chỉ bằng cách tạo ra một số nguyên tố khác là $x'_p \neq x_p$ cho phần tử x và sau đó chạy thuật toán tạo bằng chứng không phải là thành viên cho x'_p. Do đó, thuộc tính bảo mật của bộ tích lũy, như được xác định trong trò chơi bảo mật ở Hình 1, sẽ

không giữ được, vì kẻ tấn công có thể xuất ra (x, W), sao cho $x \in X$ và **VerifyNonMem**(pk, x, W, c) = 1.

Tính bảo mật của bộ tích lũy của chúng tôi bắt nguồn từ tính bảo mật của bộ tích lũy của Li và cộng sự. [36], được chứng minh là an toàn theo giả định RSA mạnh mẽ và các thuộc tính của thuật toán **Map** như đã được chứng minh trong Bổ đề 4.2 và Bổ đề 4.3.

Định lý 4.1. *Bộ tích lũy của Hình 2 được bảo mật theo Định nghĩa 1 theo giả định RSA mạnh mẽ và khả năng chống va chạm của* **Map** *trong mô hình Oracle ngẫu nhiên.*

Chứng minh. Giả sử có một kẻ tấn công ppt $\mathcal{A}$, phá vỡ tính bảo mật của bộ tích lũy trong Hình 2. Sau đó, theo Định nghĩa 1, $\mathcal{A}$ xuất ra (x^*, W^*), sao cho: (1) $x^* \notin X$ và **VerifyMem**(pk, x^*, W^*, c) = 1, hoặc (2) $x^* \in X$ và **VerifyNonMem**(pk, x^*, W^*, c) = 1. Giả sử (1) đúng. Sau đó, có hai trường hợp có thể xảy ra: (a) $\mathcal{A}$ tạo ra x, x^*, sao cho **Map**(x) = **Map**(x^*) và $x \in X$, do đó, phá vỡ khả năng chống va chạm của quy trình **Map**, hoặc (b) $\mathcal{A}$ tính toán một bằng chứng thành viên hợp lệ W^* cho số nguyên tố x_p^*, trong đó, **Map**(x^*) = x_p^* và $x^* \notin X$. Trong trường hợp sau, chúng tôi có thể xây dựng kẻ tấn công ppt $\mathcal{B}$, điều này phá vỡ giả định RSA mạnh. Hãy tham khảo để biết thêm chi tiết về bằng chứng của Li và cộng sự [36]. Tiếp theo, giả sử (2) đúng. Điều này cho thấy hai tình huống có thể xảy ra: Đầu tiên, $\mathcal{A}$ đưa ra một bằng chứng không phải thành viên hợp lệ W^* cho số nguyên tố x_p^*, trong đó, **Map**(x^*) = x_p^* và $x \in X$. Điều này nghĩa là chúng tôi có thể xây dựng một kẻ tấn công ppt $\mathcal{B}$, điều này phá vỡ giả định RSA mạnh (xem chứng minh của Li và cộng sự [36]) và do đó, chúng tôi có mâu thuẫn. Trong kịch bản thứ hai, quy trình **Map** với đầu vào x^*, xuất ra hai số nguyên tố khác nhau (với xác suất không đáng kể) nếu chúng ta chạy nó hai lần, ví dụ: $x_{p_1}^*$ và $x_{p_2}^*$. Điều này nghĩa là nếu $x_{p_1}^*$ được thêm vào bộ tích lũy trước thì $\mathcal{A}$ có thể tính được một bằng chứng không phải thành viên hợp lệ W^* cho $x_{p_2}^*$. Tuy nhiên, điều này là không thể, vì quy trình **Map** mang tính xác định. □

Thiết kế bộ tích lũy phổ quát từ bộ tích lũy bổ sung, phổ quát[13]. Giả sử ACC_U^{add} là một bộ tích lũy phổ quát, bổ sung, tích lũy các phần tử có dạng (x, i, **op**), trong đó x là phần tử được thêm vào, i là một chỉ số và **op** là a hoặc d. Chúng tôi xây dựng bộ tích lũy phổ quát ACC_U, từ ACC_U^{add}, như sau. Khi một phần tử x được thêm vào ACC_U lần đầu tiên, T_{acc} sẽ thêm giá trị (x, 1, a) vào ACC_U^{add}. Mặt khác, nó thêm (x, i, a), trong đó, chỉ số i chỉ ra rằng đây là lần thứ i x được thêm vào ACC_U^{add}. Khi một phần tử x bị xóa khỏi ACC_U, T_{acc} thêm (x, i, d) vào ACC_U^{add}. Để chứng minh tư cách thành viên của x trong ACC_U, người ta phải tìm một chỉ mục i sao cho $((x, i, a) \in ACC_U^{add}) \wedge ((x, i, d) \notin ACC_U^{add})$. Theo đó, để chứng minh rằng $x \notin X$, người ta phải chứng minh rằng $(x, 1, a) \notin ACC_U^{add}$, hoặc tìm chỉ số i, sao cho $((x, i-1, d) \in ACC_U^{add}) \wedge ((x,i,a) \notin ACC_U^{add})$.

Hàm dịch vụ đặt tên $\mathcal{F}_{ns}$:

- Máy chủ S_i với đầu vào (*sid*, **Init**), $\mathcal{F}_{ns}$ gửi (*sid*, **Init**, S_i) tới $\mathcal{S}$. Nếu $\mathcal{S}$ trả về **allow**, $\mathcal{F}_{ns}$ thiết lập $Y \leftarrow Y \cup \{S_i\}$ (trong đó Y được khởi tạo là $Y = \emptyset$) và trả về **success** cho S_i. Khi tất cả các máy chủ đã gửi thông điệp (*sid*, **Init**), $\mathcal{F}_{ns}$ thiết lập **flag** = **start**. Ngoài ra, $\mathcal{S}$ còn khiến một số máy trạm trở thành gian lận. Chúng tôi biểu thị là C_{cor} tập hợp các máy trạm gian lận.

- T với đầu vào (*sid*, **Setup**, R), $\mathcal{F}_{ns}$ kiểm tra xem **flag** = **start** và chuyển tiếp (*sid*, **Setup**, R) tới $\mathcal{S}$. Nếu $\mathcal{S}$ trả về **allow**, $\mathcal{F}_{ns}$ lưu trữ R, khởi tạo một tập hợp $X \leftarrow \emptyset$, thiết lập **flag** = **manage** và trả về **success** cho T.

- Máy tạm C với đầu vào (*sid*, **Register**, *id*, *pk*), $\mathcal{F}_{ns}$ kiểm tra xem **flag** = **manage** có đúng không và chuyển tiếp (*sid*, **Register**, *id*, *pk*) tới $\mathcal{S}$.
  - Nếu $\mathcal{S}$ trả về **allow**, $\mathcal{F}_{ns}$ kiểm tra xem $(id,\cdot) \in X$. Nếu không có $(id,\cdot) \in X$, nó thiết lập $X \leftarrow X \cup (id, pk)$ và trả về **success** cho C thông qua đầu ra có độ trễ công khai, nếu không, nó trả về **fail** cho C thông qua đầu ra có độ trễ công khai.
  - Nếu $\mathcal{S}$ trả về **fail** thì $\mathcal{F}_{ns}$ trả về **fail** cho C thông qua đầu ra có độ trễ công khai.
  - Nếu $\mathcal{S}$ trả về (*sid*, **Register**, id', pk', C), $\mathcal{F}_{ns}$ kiểm tra xem $C \in C_{cor}$ và $(id',\cdot) \in X$ có đúng không. Nếu không có $(id',\cdot) \in X$, nó thiếp lập $X \leftarrow X \cup (id', pk')$ và trả về **success** cho C thông qua đầu ra có độ trễ công khai, nếu không, nó sẽ trả về **fail** cho C thông qua đầu ra có độ trễ công khai.

- C với đầu vào (*sid*, **Revoke**, *id*, *pk*, **aux**), $\mathcal{F}_{ns}$ kiểm tra xem **flag** = **manage** và chuyển tiếp (*sid*, **Revoke**, *id*, *pk*, **aux**) tới $\mathcal{S}$.
  - Nếu $\mathcal{S}$ trả về **allow**, $\mathcal{F}_{ns}$ kiểm tra xem R(*pk*, **aux**) = 1 và (*id*, *pk*) $\in X$. Nếu cả hai điều kiện đều đúng, $\mathcal{F}_{ns}$ tính toán $X \leftarrow X \setminus$(*id*, *pk*) và trả về **success** cho C thông qua đầu ra có độ trễ công khai, nếu không, nó sẽ trả về **fail** cho C thông qua đầu ra có độ trễ công khai.
  - Nếu $\mathcal{S}$ trả về **fail** thì $\mathcal{F}_{ns}$ trả về **fail** cho C thông qua đầu ra có độ trễ công khai.
  - Nếu $\mathcal{S}$ trả về (*sid*, **Revoke**, id', pk', **aux**$'$, C), $\mathcal{F}_{ns}$ kiểm tra $C \in C_{cor}$, R(pk', **aux**$'$) = 1 và $(id', pk') \in X$. Nếu vậy, $\mathcal{F}_{ns}$ tính $X \leftarrow X \setminus (id', pk')$ và trả về **success** cho C thông qua đầu ra có độ trễ công khai, nếu không, nó trả về **fail** cho C thông qua đầu ra có độ trễ công khai.

- C với đầu vào (*sid*, **Retrieve**, *id*), nếu **flag** = **manage**, $\mathcal{F}_{ns}$ chuyển tiếp thông điệp này tới $\mathcal{S}$. Nếu $\mathcal{S}$ trả về **allow** thì nếu có một cặp (*id, pk*) $\in X$ với một vài *pk*, $\mathcal{F}_{ns}$ trả về *pk* cho C thông qua đầu ra có độ trễ công khai, nếu không, nó trả về $\perp$. Nếu $\mathcal{S}$ trả về **fail** cho $\mathcal{F}_{ns}$ thì $\mathcal{F}_{ns}$ trả về **fail** cho C thông qua đầu ra có độ trễ công khai.

- Máy trạm C với đầu vào (*sid*, **VerifyID**, *id*), nếu **flag** = **manage**, $\mathcal{F}_{ns}$ chuyển tiếp thông điệp này tới $\mathcal{S}$. Nếu $\mathcal{S}$ trả về **allow** thì nếu có một cặp (*id*, *pk*) $\in X$ với một vài *pk*, $\mathcal{F}_{ns}$ trả về 1 cho C thông qua đầu ra có độ trễ công khai, nếu không, nó trả về 0. Nếu $\mathcal{S}$ trả về **fail** cho $\mathcal{F}_{ns}$ thì $\mathcal{F}_{ns}$ trả về **fail** cho C thông qua đầu ra có độ trễ công khai.

- Máy trạm C với đầu vào (*sid*, **VerifyMapping**, *id*, *pk*), nếu **flag** = **manage**, $\mathcal{F}_{ns}$ chuyển tiếp thông điệp này tới $\mathcal{S}$. Nếu $\mathcal{S}$ trả về **allow** thì nếu có một cặp (*id*, *pk*) $\in X$, $\mathcal{F}_{ns}$ trả về 1 cho C thông qua đầu ra có độ trễ công khai, nếu không, nó trả về 0. Nếu $\mathcal{S}$ trả về **fail** cho $\mathcal{F}_{ns}$ thì $\mathcal{F}_{ns}$ trả về **fail** cho C thông qua đầu ra có độ trễ công khai.

Hình 3: Hàm dịch vụ đặt tên $\mathcal{F}_{ns}$ tương tác với một tập hợp *n* máy trạm, một tập hợp *m* máy chủ, bên đáng tin cậy T và trình mô phỏng $\mathcal{S}$. Nó cho phép các máy trạm đăng ký, thu hồi, truy xuất và xác minh ánh xạ (*id*, *pk*).

5 Xác định hàm dịch vụ đặt tên

Trong phần này, chúng tôi mô tả tính bảo mật của dịch vụ đặt tên trong khung UC ([19]) bằng cách xác định nó như một hàm lý tưởng $\mathcal{F}_{ns}$ (Hình 3). $\mathcal{F}_{ns}$ tương tác với *n* máy trạm, *m* máy chủ, bên *T* chịu trách nhiệm thiết lập và kẻ tấn công $\mathcal{S}$, được gọi là trình mô phỏng. Nó lưu trữ các cặp (danh tính, khóa công khai) và hỗ trợ một số hoạt động. Các máy chủ chịu trách nhiệm chạy dịch vụ đặt tên và do đó, trước khi thiết lập, chúng tôi yêu cầu tất cả các máy chủ gửi thông điệp (*sid*, **Init**). Trong quá trình thiết lập, bên *T* chỉ định một mối quan hệ *R* xác định theo điều kiện nào thì khóa công khai có thể bị thu hồi. Trong thực tế, mối quan hệ này có thể là một thuật toán xác minh cho bằng chứng NIZK hoặc chữ ký trên một thông điệp được chọn ngẫu nhiên. Sau giai đoạn thiết lập, máy trạm có thể đăng ký một cặp (danh tính, khóa công khai), giả sử danh tính có sẵn và có thể thu hồi cặp (danh tính, khóa công khai), giả sử khóa công khai của nó thỏa mãn mối quan hệ *R*. Hơn nữa, nó có thể truy xuất khóa công khai của danh tính đã đăng ký và kiểm tra xem danh tính hoặc cặp (danh tính, khóa công khai) đã được đăng ký hay chưa. Mô hình của chúng tôi chỉ xem xét các lỗi tĩnh, do đó, chúng tôi giả sử trình mô phỏng xác định tập hợp các máy trạm gian lận C_{cor} trước khi thiết lập. Bên *T* được coi là đáng tin cậy, do đó, trình giả lập $\mathcal{S}$ không được phép làm *T* bị lỗi. Ngoài ra, hãy lưu ý rằng, trong thực tế, hàm này không thể thực hiện được đối với bất kỳ mô hình gian lận nào của *m* máy chủ. Tuy nhiên, mô hình lỗi của máy chủ phụ thuộc vào giao thức mà chúng tôi hướng tới để hiện thực hóa hàm $\mathcal{F}_{ns}$.

6 Triển khai dịch vụ đặt tên

Ở mức cao, dịch vụ phải cho phép lưu trữ, truy xuất và xóa các cặp (danh tính, khóa công khai). Rào cản chính trong việc hiện thực hóa DPKI dựa trên hợp đồng thông minh là quy mô trạng thái của nó, vốn là nguồn tài nguyên đắt nhất để truy cập, phải được giảm thiểu. Chúng tôi giải quyết vấn đề này theo hai cách. Đầu tiên, chúng tôi tách bộ nhớ khỏi quá trình xác minh tính hợp lệ của ánh xạ bằng cách duy trì hai bộ tích lũy phổ quát, trạng thái công khai (như được trình bày trong Phần 4) làm trạng thái hợp đồng thông minh. Thứ hai, bộ tích lũy của chúng tôi dựa trên RSA và có kích thước không đổi. Thuộc tính trạng thái công khai là bắt buộc vì trạng thái của hợp đồng có thể truy cập công khai. Để tránh vấn đề việc xóa yêu cầu quyền truy cập vào khóa riêng tư của bộ tích lũy, chúng tôi sử dụng phương pháp được trình bày ở cuối Phần 4.

Trong Hình 4, chúng tôi xác định hàm $\mathcal{F}_{TP}$, hàm này thể hiện vai trò của hợp đồng thông minh trong giao thức của chúng tôi. Hàm này tương tác với một bên *T*, một tập hợp *n* máy trạm và một tập hợp *m* máy chủ, một số trong đó có thể bị kẻ tấn công làm hỏng trước giai đoạn khởi tạo. $\mathcal{F}_{TP}$ được khởi tạo bởi bên tin cậy *T* bằng cách nhận đầu vào là chương trình *P*. Trạng thái của $\mathcal{F}_{TP}$ được cập nhật sau lệnh gọi đến chương trình *P* và đầu ra được nhận bởi bên gọi. Lưu ý rằng việc triển khai $\mathcal{F}_{TP}$ yêu cầu phần lớn các máy chủ trung thực, theo lộ trình của [28, 29, 12]. Kẻ tấn công luôn có đầy đủ kiến thức về tất cả các tính toán được thực hiện và có thể can thiệp bằng cách hủy bỏ hoặc cho phép thực hiện *P* theo ý muốn. Tuy nhiên, anh ta bị hạn chế sửa đổi đầu ra. Việc triển khai $\mathcal{F}_{TP}$ bằng giao thức Blockchain có các máy chủ đóng vai trò là "thợ đào", *T* và các máy trạm tương tác với Blockchain bằng cách đăng các giao dịch. Cài đặt chương trình *P* là một giao dịch đặc biệt bao gồm *P* trong Blockchain và sau đó, việc thực thi P yêu cầu tất cả các thợ đào phải chạy nó và ghi lại cập

nhật trạng thái của nó trong Blockchain. Các thuộc tính bảo mật của Blockchain cơ bản, đặc biệt liên quan đến sự tồn tại của các giao dịch, xem [28, 29, 12], cho thấy tính bảo mật của việc thực hiện $\mathcal{F}_{TP}$.

$\mathcal{F}_{TP}$:

- Máy chủ S_i với đầu vào (*sid*, **InitTP**), $\mathcal{F}_{TP}$ thiết lập $S_{init} \leftarrow S_{init} \cup \{S_i\}$ (khởi tạo là $S_{init} \leftarrow \emptyset$) và thông báo cho kẻ tấn công $\mathcal{A}$ rằng S_i được khởi tạo bằng cách gửi (*sid*, **InitTP**, S_i). Sau đó $\mathcal{F}_{TP}$ trả về **success** cho S_i. Nếu tất cả các máy chủ S_1,..., S_m đã gửi thông điệp (*sid*, **InitTP**), $\mathcal{F}_{TP}$ thiết lập **flag** = **ready**.
- *T* với đầu vào (*sid*, **Install**, *P*), nếu **flag** = **ready**, gửi (*sid*, **Install**, *P*) tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì thiết lập **flag** = **start**, lưu trữ *P*, thiết lập *state* ← ε, trong đó ε là chuỗi trống và trả về **success** cho *T*.
- Máy trạm *C* với đầu vào (*sid*, *x*), nếu **flag** = **ready**, gửi (*sid*, *x*) tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì chạy *P* với đầu vào (*x*, *state*) và đầu ra (*y*, $state'$). Thiết lập *state* ← $state'$ và trả về (*y*, $state'$) cho *C* thông qua đầu ra có độ trễ công khai. Nếu *x* là đầu vào không hợp lệ cho chương trình *P* thì trả về $\perp$ cho *C* thông qua đầu ra có độ trễ công khai.

Hình 4: Hàm $\mathcal{F}_{TP}$ thể hiện vai trò của hợp đồng thông minh. Nó tương tác với một bên đáng tin cậy *T*, một tập hợp *n* máy tạm, một tập hợp *m* máy chủ và kẻ tấn công $\mathcal{A}$.

1. Với đầu vào (**Setup**, *params*), trong đó *params* có dạng (pk_1, pk_2, *R*), chạy **InitAcc** với đầu vào pk_1 và pk_2, và tính $c_{0,1}$, $c_{0,2}$ tương ứng. Quy trình này khởi tạo hai bộ tích lũy phổ quát, bổ sung, trạng thái công khai c_1 và c_2 bằng cách thiết lập $c_1 \leftarrow c_{0,1}$, $c_2 \leftarrow c_{0,2}$. Nó thiết lập *state* ← (*params*, c_1, c_2) và trả về *state*.

2. Với đầu vào (**Register**, *id*, *pk*, *i*, W_1, W_2),

   (a) Nếu $i = 1$ thì kiểm tra xem **VerifyNonMem**(pk_2, (*id*, 1, *a*), W_1, c_2) = 1 có đúng không.

   (b) Nếu $i \geq 2$ thì kiểm tra xem **VerifyNonMem**(pk_2, (*id*, *i*, *a*), W_1, c_2) = 1 và **VerifyMem**(pk_2, (*id*, $i - 1$, *d*), W_2, c_2) = 1 có đúng không.

   Nếu tất cả các kiểm tra trên thành công thì chạy $(c_1', W_1') \leftarrow$ **Add**(pk_1, (*id*, *pk*, *i*, *a*), c_1) và $(c_2', W_2') \leftarrow$ **Add**(pk_2, (*id*, *i*, *a*), c_2). Cập nhật *state* bằng cách thiết lập $c_1 \leftarrow c_1'$ và $c_2 \leftarrow c_2'$, và trả về $((c_1', W_1'), (c_2', W_2'))$. Nếu không thì trả về **fail**.

3. Với đầu vào (**Revoke**, *id*, *pk*, *i*, W_1, W_2, W_3, **aux**),

   (a) Kiểm tra xem *R*(*pk*, **aux**) = 1 có đúng không.

   (b) Kiểm tra xem **VerifyMem**(pk_2, (*id*, *i*, *a*), W_1, c_2) = 1, **VerifyNonMem**(pk_2, (*id*, *i*, *d*), W_2, c_2) = 1 và **VerifyMem**(pk_1, (*id*, *pk*, *i*, *a*), W_3, c_1) = 1.

   Nếu không có xác minh nào ở trên không thành công thì chạy $(c_2', W_2') \leftarrow$ **Add**(pk_2, (*id*, *i*, *d*), c_2) và $(c_1', W_1') \leftarrow$ **Add**(pk_1, (*id*, *pk*, *i*, *d*), c_1). Cập nhật *state* bằng cách thiết lập $c_1 \leftarrow c_1'$ và $c_2 \leftarrow c_2'$, và trả về $((c_1', W_1'), (c_2', W_2'))$. Nếu không thì trả về **fail**.

4. Với đầu vào **RetrieveState**, trả về *state* ← (*params*, c_1, c_2).

Hình 5: Chương trình *P* là đầu vào của $\mathcal{F}_{TP}$, trong quá trình khởi tạo trong quá trình thiết lập của chúng tôi.

$\mathcal{F}_{UDB}$:

- Máy chủ S_i với đầu vào (*sid*, **InitUDB**), $\mathcal{F}_{UDB}$ thiết lập $S'_{init} \leftarrow S'_{init} \cup \{S_i\}$ (khởi tạo là $S'_{init} \leftarrow \emptyset$) và gửi (*sid*, **InitUDB**, S_i) đến $\mathcal{A}$. Sau đó $\mathcal{F}_{UDB}$ trả về **success** cho S_i. Nếu tất cả các máy chủ S_1,..., S_m đã gửi thông điệp (*sid*, **InitUDB**), $\mathcal{F}_{UDB}$ thiết lập **flag** = **ready**, *DBstate* ← ∅ và $p \leftarrow 0$.
- Máy trạm *C* với đầu vào (*sid*, **Post**, *x*), nếu **flag** = **ready**, chuyển tiếp (*sid*, **Post**, *x*, *C*) tới $\mathcal{A}$. Nếu $\mathcal{A}$ gửi **allow** thì thiết lập $p \leftarrow p + 1$, *DBstate* [*p*] ← *x* và trả về **success** cho *C*.
- Máy trạm *C* với đầu vào (*sid*, **RetrieveDB**), nếu **flag** = **ready**, chuyển tiếp (*sid*, **RetrieveDB**, *C*) tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì trả về *DBstate* về *C*.
- $\mathcal{A}$ với đầu vào (*sid*, **ChangeDBstate**, $DBstate'$), thiết lập *DBstate* ← $DBstate'$.

Hình 6: Hàm $\mathcal{F}_{UDB}$ mô hình hóa một cơ sở dữ liệu không đáng tin cậy và tương tác với một tập hợp *n* máy trạm, một tập hợp *m* máy chủ và kẻ tấn công $\mathcal{A}$.

Hơn nữa, chúng tôi giả định tất cả các hoạt động được hoàn thành theo kiểu nguyên tử, đồng bộ. Trong thực tế, cần một khoảng thời gian để một hoạt động (giao dịch) được xác thực, tức là được ghi lại trên Blockchain. Tuy nhiên, các Blockchain thực thi tổng các giao dịch được yêu cầu và thực hiện chúng một cách tuần tự, điều này mang lại kết quả chung như nhau. Trong giao thức của chúng tôi, $\mathcal{F}_{TP}$ là đầu vào chương trình *P* của Hình 5, do đó, $\mathcal{F}_{TP}$ về cơ bản duy trì các bộ tích lũy nói trên ở trạng thái của nó, tức là nó hoạt động như trình quản lý bộ tích lũy T_{acc}. Để đơn giản hóa mô tả và phân tích bảo mật trong thiết kế, chúng tôi giả sử giai đoạn thiết lập đáng tin cậy sẽ thiết lập mối quan hệ *R* và tạo khóa của bộ tích lũy. Giả định này không đưa ra một điểm lỗi nào trong thiết kế của chúng tôi vì nó có thể được thay thế, trong triển khai thực tế bằng các giao thức phân tán để tạo tham số (ví dụ: [46]).

Trong Hình 6, chúng tôi giới thiệu hàm $\mathcal{F}_{UDB}$, hàm này xử lý việc lưu trữ thông tin có liên quan đến giao thức, ví dụ: các cặp (danh tính, khóa công khai). $\mathcal{F}_{UDB}$ tương tác với *n* máy trạm, một tập hợp ℓ máy chủ và kẻ tấn công. Hàm này mô hình hóa một "cơ sở dữ liệu không đáng tin cậy", tức là kẻ tấn công có thể giả mạo nội dung của nó. Có hai điều về sự tham gia của nó vào giao thức của chúng tôi. Đầu tiên, máy trạm truy vấn hàm này để lấy tất cả thông tin cần thiết cho phép nó tương tác với $\mathcal{F}_{TP}$ sau đó. Thứ hai, sau khi hoàn thành tương tác với F_{TP}, máy trạm sẽ lưu trữ trong $\mathcal{F}_{UDB}$ cùng với những thông tin khác được hợp đồng thông minh xuất ra và phản ánh trạng thái mới của hệ thống. Chúng tôi sẽ trình bày chi tiết hơn về thông tin mà máy trạm truy vấn/lưu trữ từ/đến $\mathcal{F}_{UDB}$ sau trong phần này, ở đó chúng tôi cung cấp mô tả cấp cao về từng hoạt động. Việc thể hiện thực tế $\mathcal{F}_{UDB}$ nằm ngoài phạm vi của bài viết này. Tuy nhiên, mạng lưới DHT được xác thực bao gồm các Node đã đăng ký trong PKI của chúng tôi sẽ là một ứng viên phù hợp, cả về mặt bảo mật (tức là nó có khả năng phục hồi Sybil), cũng như hiệu quả, do độ phức tạp của thông điệp Logarit của nó.

Trong sơ đồ của chúng tôi, chúng tôi tích lũy các bộ dữ liệu (*id*, *pk*, *i*, **op**) trong c_1, trong đó, **op** = *a* hoặc **op** = *d* đánh dấu một phần tử lần lượt là "đã thêm" hoặc "đã xóa". Điều này cho phép máy trạm suy ra ánh xạ (danh tính, khóa công khai) có hợp lệ hay không. Trong c_2, chúng tôi tích lũy các bộ dữ liệu (*id*, *i*, **op**), cho phép máy trạm suy ra danh tính có được đăng ký trong hệ thống hay không. Trong phần sau đây và do giới hạn về không gian, chúng tôi trình bày mô tả cấp cao về các hoạt động **Register**, **Revoke**, **Retrieve**, **VerifyID** và **VerifyMapping**.

Một cách không chính thức, một máy trạm quan tâm đến việc đăng ký ánh xạ (danh tính, khóa công khai) phải chứng minh với hợp đồng thông minh rằng danh tính hiện có sẵn. Để đạt được điều này, nó tạo ra hai bằng chứng. Đầu tiên, một bằng chứng thành viên của bộ dữ liệu (*id*, *i*, *d*) cho c_2, chứng tỏ rằng phiên bản thứ *i* của danh tính này đã được đánh dấu là đã xóa. Thứ hai, một bằng chứng không phải là thành viên của bộ dữ liệu (*id*, *i* + 1, *a*) cho c_2, chứng tỏ rằng phiên bản thứ (*i* + 1) của danh tính này, phiên bản mà nó quan tâm đến việc đăng ký, không được đánh dấu là đã thêm. Nếu cả hai điều kiện nói trên đều được giữ nguyên, nó có thể thuyết phục hợp đồng thông minh tích lũy ánh xạ của mình trong c_1. Những bằng chứng này được xây dựng bằng cách, trước tiên, truy vấn F_{UDB} về lịch sử hoạt động và thứ hai, định vị các bản ghi liên quan đến *id*, trong nỗ lực tìm giá trị thích hợp cho chỉ số *i*. Sau khi đăng ký thành công, máy trạm đăng một bản ghi (**Register**, *id*, *pk*, *i*, W_1, W_2, W_3) đến F_{UDB}. Các bằng chứng W_1, W_2, W_3 tạo điều kiện thuận lợi cho các truy vấn từ náy trạm trong tương lai, ví dụ như về tính hợp lệ của phiên bản thứ (*i* +1) của ánh xạ (danh tính, khóa công khai) của nó.

Để thu hồi ánh xạ (danh tính, khóa công khai), máy trạm tạo ra các bằng chứng sau. Đầu tiên, bằng chứng về quyền sở hữu khóa bí mật tương ứng được nắm bắt bởi mối quan hệ *R*. Thứ hai, một bằng chứng thành viên của bộ dữ liệu (*id*, *i*, *a*) cho c_2, chứng tỏ rằng danh tính này được đánh dấu là đã thêm cho chỉ số *i*. Thứ ba, một bằng chứng không phải là thành viên của bộ dữ liệu (*id*, *i*, *d*) cho c_2, chứng tỏ rằng danh tính này chưa được đánh dấu là đã xóa đối với chỉ số *i*. Thứ tư, một bằng chứng thành viên của bộ dữ liệu (*id*, *pk*, *i*, *a*) cho c_1, chứng minh rằng danh tính thực sự được ánh xạ tới cùng một khóa công khai thỏa mãn mối quan hệ *R*. Giả sử các bằng chứng được tạo ra một cách trung thực, máy trạm sẽ thuyết phục hợp đồng thu hồi ánh xạ và sau đó, nó tiếp tục đăng (**Revoke**, *id*, *pk*, *i*) đến F_{UDB}.

Để truy xuất khóa công khai của danh tính, máy trạm truy vấn F_{UDB} để kiểm tra xem bản ghi cuối cùng liên quan đến danh tính này có phải là bản ghi đăng ký hay không. Nếu vậy, máy trạm sẽ cập nhật các bằng chứng W_1, W_2, W_3 được lưu trữ trong bản ghi đăng ký được truy xuất và sau đó, gọi hợp đồng thông minh để xác thực ánh xạ (danh tính, khóa công khai). **VerifyID** và **VerifyMapping** tuân theo quy trình tương tự như **Retrieve** để xác minh xem danh tính hoặc ánh xạ (danh tính, khóa công khai) đã được đăng ký hay chưa.

Trong Hình 7, chúng tôi trình bày mô tả chính thức của giao thức π, thực hiện hàm $\mathcal{F}_{ns}$. Hãy nhớ lại rằng các thực thể tham gia vào giao thức là: 1) bên đáng tin cậy T được sử dụng cho mục đích thiết lập, 2) hàm $\mathcal{F}_{TP}$, 3) *n* máy trạm C_1,...,C_n và, 4) hàm $\mathcal{F}_{UDB}$. Chúng tôi ký hiệu X_1 và X_2 lần lượt là tập hợp các phần tử tích lũy của c_1 và c_2. Những tập hợp này được máy trạm thiết kế như sau. Đối với bất kỳ bản ghi nào có dạng (**Register**, *id*, *pk*, *i*,·), máy trạm thêm (*id*, *pk*, *i*, *a*) vào X_1 và (*id*, *i*, *a*) vào X_2. Đối với bất kỳ bản ghi nào có dạng (**Revoke**, *id*, *pk*, *i*), máy trạm thêm (*id*, *pk*, *i*, *d*) vào X_1 và (*id*, *i*, *d*) vào X_2.

Để dễ trình bày, chúng tôi đã mô tả giao thức bằng cách sử dụng hai bộ tích lũy. Chúng tôi có thể đạt được kết quả cuối cùng giống nhau chỉ bằng cách sử dụng một bộ tích lũy vì cả c_1 và c_2 đều tích lũy các chuỗi tùy ý. Do đó, chúng tôi có thể tích lũy cả hai loại bộ dữ liệu, tức là (*id*, *i*, **op**) và (*id*, *pk*, *i*, **op**), trong một bộ tích lũy, trong khi vẫn có thể tạo ra các bằng chứng thành viên và không phải thành viên được yêu cầu trong giao thức của chúng tôi. Để đạt được điều này, chúng tôi sửa đổi các hoạt động **Register** và **Revoke** của chương trình *P* như sau. Đầu tiên, lệnh gọi thứ hai tới **Add**, trong cả hai thao tác, sẽ nhận dưới dạng tham số giá trị tích lũy được trả về từ lệnh gọi đầu tiên tới **Add**.

1. Với đầu vào (*sid*, **Init**), máy chủ S_i gửi (*sid*, **InitTP**) và (*sid*, **InitUDB**) đến $\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$. Nếu S_i nhận được **success** bởi cả $\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$ thì S_i trả về **success**.
2. Với đầu vào (*sid*, **Setup**, *R*), bên *T* gửi (*sid*, **Install**, *P*) tới $\mathcal{F}_{TP}$, trong đó *P* là chương trình của Hình 5. Nếu $\mathcal{F}_{TP}$ trả về **success** thì *T* chạy **KeyGen**(1^λ) hai lần, thiết lập *params* = (pk_1, pk_2, *R*) và gửi (*sid*, (**Setup**, *params*)) tới $\mathcal{F}_{TP}$, thực thi chương trình *P* với đầu vào (**Setup**, *params*) (nếu $\mathcal{A}$ trả về **allow** tới $\mathcal{F}_{TP}$). Do đó, nếu $\mathcal{F}_{TP}$ trả về *state* ← (*params*, c_1, c_2) cho *T* thì *T* trả về **success**.
3. Với đầu vào (*sid*, **Register**, *id*, *pk*), *C* gửi (*sid*, **RetrieveDB**) tới $\mathcal{F}_{UDB}$. Khi nhận được *DBstate*, *C* sẽ kiểm tra các bản ghi liên quan đến *id*.
   (a) Nó tìm bản ghi cuối cùng liên quan đến *id* và kiểm tra xem nó có ở dạng (**Revoke**, *id*, *pk*, *i*) không. Nếu không thì *C* trả về **fail**. Mặt khác, nó tính toán bằng chứng không phải là thành viên W_1 cho (*id*, *i* + 1, *a*) và bằng chứng thành viên W_2 cho (*id*, *i*, *d*) bằng cách chạy **NonMemWitGen**(pk_2, (*id*, *i* + 1, *a*), X_2, c_2) và **MemWitGen**(pk_2, (*id*, *i*, *d*), X_2, c_2) tương ứng.
   (b) Nếu không tìm thấy bản ghi nào, *C* tính toán bằng chứng không phải là thành viên W_1 cho (*id*, 1, *a*) bằng cách chạy **NonMemWitGen**(pk_2, (*id*, 1, *a*), X_2, c_2), thiếp lập $W_2 = \perp$ và $i = 0$.

   Sau đó, *C* gửi (*sid*, **Register**, *id*, *pk*, *i* + 1, W_1, W_2) đến $\mathcal{F}_{TP}$, chạy *P* trên đầu vào này. Nếu $\mathcal{F}_{TP}$ trả về $((c_1', W_1'), (c_2', W_2'), state)$, trong đó W_1' là bằng chứng thành viên cho (*id*, *pk*, *i* + 1, *a*) trong c_1' và W_2' là bằng chứng thành viên cho (*id*, *i* + 1, *a*) trong c_2'. *C* đưa ra một bằng chứng không phải là thành viên W_3' cho (*id*, *i* + 1, *d*) bằng cách chạy **NonMemWitGen**((*id*, *i* + 1, *d*), X_2, c_2') và gửi (*sid*, **Post**, (**Register**, *id*, *pk*, *i* + 1, W_1', W_2', W_3') đến F_{UDB}. Nếu F_{UDB} trả về **success** thì *C* xuất ra **success**. Nếu không thì *C* xuất ra **fail**.
4. Với đầu vào (*sid*, **Revoke**, *id*, *pk*, **aux**), *C* gửi (*sid*, **RetrieveDB**) tới $\mathcal{F}_{UDB}$. Khi nhận được *DBstate*, *C* tìm kiếm các bản ghi trước bản ghi đăng ký của mình. Giả sử ℓ bản ghi như vậy và tùy thuộc vào bản ghi gặp phải có dạng **Register**, hoặc, **Revoke**, *C* trên mỗi lần lặp, thì cập nhật nhân chứng của nó như sau :
   (a) *C* gặp a (**Register**, $id', pk', j, W_1^{id'}, W_2^{id'}, W_3^{id'}$). Nó cập nhật W_1' bằng cách chạy $W_1' \leftarrow$ **UpdMemWit**(pk_1, (*id*, *pk*, *i*, *a*), (id', pk', *j*, *a*), W_1'). W_2', W_3' được cập nhật tương ứng.
   (b) *C* gặp (**Revoke**, id', pk', *j*). Nó cập nhật W_1' bằng cách chạy $W_1' \leftarrow$ **UpdMemWit**(pk_1, (*id*, *pk*, *i*, *a*), (id', pk', *j*, *d*), W_1'). W_2', W_3' được cập nhật tương ứng.

   Sau đó, *C* gửi (*sid*, **Revoke**, *id*, *pk*, *i*, W_1'). W_2', W_3', **aux**) tới $\mathcal{F}_{TP}$. Nếu $\mathcal{F}_{TP}$ trả về $((c_1', W_1'), (c_2', W_2'), state)$, *C* gửi (*sid*, **Post**, (**Revoke**, *id*, *pk*, *i*) đến $\mathcal{F}_{UDB}$. Nếu $\mathcal{F}_{UDB}$ trả về **succes** thì *C* xuất ra **success**. Nếu không thì *C* xuất ra **fail**.
5. Với đầu vào (*sid*, **Retrieve**, *id*), *C* gửi (*sid*, **RetrieveDB**) tới $\mathcal{F}_{UDB}$. Nếu không có bản ghi liên quan đến *id* thì *C* xuất ra **fail**, nếu không thì xuất ra *C*:
   (a) Kiểm tra xem bản ghi cuối cùng liên quan đến *id* có dạng **Register** hay không. Nếu vậy, nó truy xuất W_1', W_2', W_3' từ bản ghi, chạy Bước 4a và 4b để tính toán các bằng chứng được cập nhật W_1', W_2', W_3'. Nếu không thì *C* xuất ra **fail**.
   (b) Gửi (*sid*, **RetrieveState**) tới $\mathcal{F}_{TP}$. Nếu $\mathcal{F}_{TP}$ trả về *state* thì *C* chạy **VerifyMem**(pk_1, (*id*, *pk*, *i*, *a*), W_1', c_1), **VerifyMem**(pk_2, (*id*, *i*, *a*), W_2', c_2) và **VerifyNonMem**(pk_2, (*id*, *i*, *d*), W_3', c_2). Nếu tất cả các thuật toán xuất ra 1 thì *C* xuất ra *pk* dưới dạng khóa công khai được truy xuất, nếu không thì *C* xuất ra $\perp$.
6. Với đầu vào (*sid*, **VerifyID**, *id*), *C* chạy Bước 5. Nếu Bước 5 xuất ra một vài *pk* thì *C* xuất ra 1, nếu không thì *C* xuất ra 0.
7. Với đầu vào (*sid*, **VerifyMapping**, *id*, *pk*), *C* chạy Bước 5. Nếu Bước 5 xuất ra *pk* thì *C* xuất ra 1, nếu không thì *C* xuất ra 0.

Hình 7: Mô tả giao thức π được xây dựng dựa trên chương trình *P* của Hình 5.

Thứ hai, chúng tôi gọi **UpdMemWit** sau lệnh gọi thứ hai tới **Add**, để cập nhật bằng chứng tư cách thành viên được trả về bởi lệnh gọi đầu tiên tới **Add**. Cách tiếp cận này cắt giảm một nửa trạng thái của hợp đồng, nhưng lại tăng tính toán của cả **Register** và **Revoke** bằng một phép lũy thừa và một lệnh gọi quy trình **Map**.

Cuối cùng, chúng tôi chứng minh thiết kế là an toàn bằng cách chứng minh Định lý 6.1. Chứng minh Định lý 6.1 được thể hiện trong Phụ lục A.

Định lý 6.1. *Giao thức π của Hình 7 thực hiện một cách an toàn hàm $\mathcal{F}_{ns}$ của Hình 3 trong thế giới hỗn hợp ($\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$) theo giả định RSA mạnh trong mô hình Oracle ngẫu nhiên.*

Tài liệu tham khảo

[1] Ascap, prs và sacem hợp tác vì hệ thống bản quyền Blockchain. https:// www.musicbusinessworldwide.com/ascap-prs-sacem-join-forces-Blockchain-Copyright-system/. Truy cập: 2017-07-06.

[2] Emercoin - dịch vụ Blockchain phân tán cho doanh nghiệp và sử dụng cá nhân. http://www.emercoin.com. Truy cập: 2010-09-30.

[3] Báo cáo cuối cùng về vụ hack diginotar cho thấy sự xâm nhập hoàn toàn của các máy chủ ca. https://threatpost.com/final-report-diginotar-hack-shows-total-compromiseca-servers-103112/77170/. Truy cập: 2017-04-07.

[4] Google đưa symantec đến phòng xử lý gỗ vì phát hành sai 30.000 chứng chỉ https. https://arstechnica.com/information-technology/2017/03/google-takes-symantec-to-the-woodshed-for-mis-issuing-30000-https-certs/. Truy cập: 2017-04-07.

[5] IBM đẩy Blockchain vào chuỗi cung ứng. https://www.wsj.com/articles/ibm-push-Blockchain-into-the-supply-chain-1468528824. Truy cập: 2017-07-06.

[6] Namecoin. https://namecoin.org/. Truy cập: 2017-04-07.

[7] Tập đoàn công nghiệp Thụy Sĩ sử dụng Blockchain Ethereum. https://www.ccn.com/swiss-industry-consortium-use-ethereums-Blockchain/. Truy cập: 2017-07-06.

[8] Trustwave thừa nhận họ đã cấp chứng chỉ cho phép công ty thực hiện các cuộc tấn công trung gian. https://www.techdirt.com/articles/20120208/03043317695/trustwave-admits-it-issued-certificate-to-allow-company-to-run-man-in-the-middle-attacks.shtml. Truy cập: 2017-04-07.

[9] Karl Aberer. *P-Grid: Cấu trúc truy cập tự tổ chức cho hệ thống thông tin P2P*. Springer Berlin Heidelberg, 2001.

[10] Man Ho Au, Patrick P. Tsang, Willy Susilo và Yi Mu. Bộ tích lũy phổ quát động cho các nhóm DDH và ứng dụng của chúng vào các hệ thống thông tin xác thực ẩn danh dựa trên thuộc tính. Trong Marc Fischlin, biên tập viên, *Chủ đề về Mật mã học - CT-RSA 2009, Đường đi của các nhà mật mã học tại Hội nghị RSA 2009, San Francisco, CA, Hoa Kỳ, ngày 20-24 tháng 4 năm 2009. Kỷ yếu*, tập 5473 của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 295–308. Springer, 2009.

[11] Agapios Avramidis, Panayiotis Kotzanikolaou, Christos Douligeris và Mike Burmester. Chord-pki: Cơ sở hạ tầng tin cậy được phân phối dựa trên mạng p2p. *Mạng máy tính*, 56, 2012.

[12] Christian Badertscher, Ueli Maurer, Daniel Tschudi và Vassilis Zikas. Bitcoin như một sổ cái giao dịch: Một phương pháp xử lý tổng hợp. Trong Katz và Shacham [34], trang 324–356.

[13] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin và Sophia Yokoubov. Bộ tích lũy có ứng dụng thu hồi bảo toàn tính ẩn danh. Trong *Hội nghị chuyên đề Châu Âu của IEEE về Bảo mật và Quyền riêng tư năm 2017, EuroS&P 2017, Paris, Pháp, ngày 26-28 tháng 4 năm 2017*, trang 301–315. IEEE, 2017.

[14] Niko Bari và Birgit Pfitzmann. Bộ tích lũy không va chạm và sơ đồ chữ ký không cây và không bị lỗi dừng. Trong Walter Fumy, biên tập viên, *Những tiến bộ trong mật mã học - EUROCRYPT '97, Hội nghị quốc tế về lý thuyết và ứng dụng kỹ thuật mật mã, Konstanz, Đức, ngày 11-15 tháng 5 năm 1997, Kỷ yếu, tập 1233* của *Ghi chú bài giảng về Khoa học máy tính*, trang 480– 494. Springer, 1997.

[15] Josh Cohen Benaloh và Michael de Mare. Bộ tích lũy một chiều: Một giải pháp thay thế phi tập trung cho các hình sin kỹ thuật số (tóm tắt mở rộng). Trong Tor Helleseth, biên tập viên, *Những tiến bộ trong mật mã học - EUROCRYPT '93, Hội thảo về lý thuyết và ứng dụng kỹ thuật mật mã, Lofthus, Na Uy, ngày 23-27 tháng 5 năm 1993, Kỷ yếu, tập 765* của *Ghi chú bài giảng về Khoa học máy tính*, trang 274– 285. Springer, 1993.

[16] Philippe Camacho, Alejandro Hevia, Marcos A. Kiwi và Roberto Opazo. Bộ tích lũy mạnh từ Hash chống va chạm. Trong Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, và Der-Tsai Lee, biên tập viên, *An toàn thông tin, Hội nghị quốc tế lần thứ 11, ISC 2008, Đài Bắc, Đài Loan, ngày 15-18 tháng 9 năm 2008. Kỷ yếu* , tập 5222 của *Ghi chú Bài giảng trong Khoa học Máy tính*, trang 471–486. Mùa xuân, 2008.

[17] Jan Camenisch, Markulf Kohlweiss và Claudio Soriente. Công cụ tích lũy dựa trên bản đồ song tuyến tính và thu hồi hiệu quả thông tin xác thực ẩn danh. Trong Stanislaw Jarecki và Gene Tsudik, các biên tập viên, *Mật mã khóa công khai - PKC 2009, Hội nghị quốc tế lần thứ 12 về thực hành và lý thuyết về mật mã khóa công khai, Irvine, CA, Hoa Kỳ, ngày 18-20 tháng 3 năm 2009. Kỷ yếu, tập 5443* của *Ghi chú bài giảng trong máy tính Khoa học*, trang 481–500. Springer, 2009.

[18] Jan Camenisch và Anna Lysyanskaya. Bộ tích lũy động và ứng dụng để thu hồi hiệu quả thông tin xác thực ẩn danh. Trong Moti Yung, biên tập viên, *Những tiến bộ trong mật mã học - CRYPTO 2002, Hội nghị mật mã quốc tế thường niên lần thứ 22, Santa Barbara, California, Hoa Kỳ, ngày 18-22 tháng 8 năm 2002, Kỷ yếu, tập 2442* của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 61–76. Springer, 2002.

[19] Ran Canetti. Bảo mật tổng hợp toàn cầu: Một mô hình mới cho các giao thức mật mã. *Lưu trữ ePrint mật mã IACR* , 2000:67, 2000.

[20] Ran Canetti. Chữ ký, chứng nhận và xác thực có thể tổng hợp toàn cầu. Trong *Hội thảo Cơ sở An ninh Máy tính IEEE lần thứ 17, (CSFW-17 2004), 28-30 tháng 6 năm 2004, Pacific Grove, CA, USA*, trang 219. Hiệp hội Máy tính IEEE, 2004.

[21] Larry Carter và Mark N. Wegman. Các lớp phổ biến của hàm băm. *J. Máy tính. Hệ thống. Khoa học.*, 18(2):143–154, 1979.

[22] Ivan Damgard và Nikos Triandopoulos. Hỗ trợ bằng chứng không phải thành viên với bộ tích lũy bản đồ song tuyến tính. *Lưu trữ ePrint mật mã IACR*, 2008:538, 2008.

[23] Anwitaman Datta, Manfred Hauswirth và Karl Aberer. Ngoài "web tin cậy": Kích hoạt thương mại điện tử P2P. Trong *CEC 2003*, trang 303–312.

[24] David Derler, Christian Hanser và Daniel Slamanig. Xem lại các bộ tích lũy mật mã, các thuộc tính bổ sung và các mối quan hệ với các dạng nguyên thủy khác. Trong Kaisa Nyberg, biên tập viên, *Chủ đề về Mật mã học - CT-RSA 2015, Bài hát của Nhà mật mã học tại Hội nghị RSA 2015, San Francisco, CA, Hoa Kỳ, ngày 20-24 tháng 4 năm 2015. Kỷ yếu, tập 9048 của Ghi chú Bài giảng về Khoa học Máy tính, các* trang 127–144. Springer, 2015.

[25] John R. Douceur. Cuộc tấn công sybil. IPTPS'01.

[26] Carl Ellison và Bruce Schneier. Mười rủi ro của PKI: Những điều bạn chưa được biết về cơ sở hạ tầng khóa công khai. 2000.

[27] Conner Fromknecht, Dragos Velicanu và Sophia Yokoubov. Cơ sở hạ tầng khóa công khai phi tập trung có khả năng lưu giữ danh tính. *Lưu trữ ePrint mật mã IACR*, 2014:803, 2014.

[28] Juan A. Garay, Aggelos Kiayias và Nikos Leonardos. Giao thức hệ thống nền tảng Bitcoin: Phân tích và ứng dụng. Trong Elisabeth Oswald và Marc Fischlin, biên tập viên, *EUROCRYPT*, tập 9057 của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 281–310. Springer, 2015.

[29] Juan A. Garay, Aggelos Kiayias và Nikos Leonardos. Giao thức hệ thống nền tảng Bitcoin với các chuỗi có độ khó khác nhau. Trong Katz và Shacham [34], trang 291–323.

[30] Rosario Gennaro, Shai Halevi và Tal Rabin. Bảo mật chữ ký Hash và ký mà không cần đến Oracle ngẫu nhiên. Trong Jacques Stern, biên tập viên, *Những tiến bộ trong mật mã học - EUROCRYPT '99, Hội nghị quốc tế về lý thuyết và ứng dụng kỹ thuật mật mã, Praha, Cộng hòa Séc, ngày 2-6 tháng 5 năm 1999, Kỷ yếu, tập 1592* của *Ghi chú bài giảng về Khoa học máy tính*, trang 123 –139. Springer, 1999.

[31] Bela Gipp, Norman Meuschke và Andre Gernandt. Đánh dấu thời gian đáng tin cậy phi tập trung bằng cách sử dụng tiền mã hoá Bitcoin. *CoRR*, abs/1502.04015, 2015.

[32] Mahabir Prasad Jhanwar và Reihaneh Safavi-Naini. Bộ tích lũy nhỏ gọn sử dụng lưới. Trong Rajat Subhra Chakraborty, Peter Schwabe và Jon A. Solworth, các biên tập viên, *Bảo mật, Quyền riêng tư và Kỹ thuật Mật mã Ứng dụng - Hội nghị Quốc tế lần thứ 5, SPACE 2015, Jaipur, Ấn Độ, ngày 3-7 tháng 10 năm 2015, Kỷ yếu, tập* 9354 của *Ghi chú Bài giảng trong Khoa học Máy tính*, trang 347–358. Springer, 2015.

[33] Murat Karakaya, Ibrahim Korpeoglu và Ozgur Ulusoy. Free-riding trong mạng ngang hàng. *Điện toán Internet IEEE*, 13(2).

[34] Jonathan Katz và Hovav Shacham, biên tập viên. *Những tiến bộ trong mật mã học - CRYPTO 2017 - Hội nghị mật mã quốc tế thường niên lần thứ 37, Santa Barbara, CA, Hoa Kỳ, ngày 20-24 tháng 8 năm 2017, Kỷ yếu, Phần I, tập 10401* của *Ghi chú Bài giảng về Khoa học Máy tính*. Springer, 2017.

[35] F. Lesueur, L. Me, và VVT Tong. Một PKI phân tán hiệu quả cho mạng P2P có cấu trúc. Trong *IEEE P2PC*, 2009.

[36] Jiangtao Li, Ninghui Li và Rui Xue. Bộ tích lũy phổ quát với bằng chứng không phải là thành viên hiệu quả. Trong Jonathan Katz và Moti Yung, biên tập viên, *Mật mã ứng dụng và An ninh mạng, Hội nghị quốc tế lần thứ 5, ACNS 2007, Chu Hải, Trung Quốc, ngày 5-8 tháng 6 năm 2007, Kỷ yếu, tập 4521* của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 253–269. Springer, 2007.

[37] Atefeh Mashatan và Serge Vaudenay. Một bộ tích lũy phổ quát hoàn toàn động. *Kỷ yếu của Học viện Rumani*, 14:269–285, 2013.

[38] Petar Maymounkov và David Mazieres. Kademlia: Hệ thống thông tin ngang hàng dựa trên số liệu xor. IPTPS '01.

[39] Satoshi Nakamoto. Bitcoin: Một hệ thống tiền điện tử ngang hàng. http://bitcoin.org/bitcoin.pdf. Truy cập: 2017-04-07.

[40] Lan Nguyen. Bộ tích lũy từ các cặp song tuyến và ứng dụng. Trong Alfred Menezes, biên tập viên, *Chủ đề về Mật mã học - CT-RSA 2005, Đường đi của các nhà mật mã học tại Hội nghị RSA 2005, San Francisco, CA, Hoa Kỳ, ngày 14-18 tháng 2 năm 2005, Kỷ yếu, tập 3376 của Ghi chú Bài giảng* về *Khoa học Máy tính*, trang 275–292. Springer, 2005.

[41] Kaisa Nyberg. Hash ích lũy nhanh. Trong Dieter Gollmann, biên tập viên, *Mã hóa phần mềm nhanh, Hội thảo quốc tế lần thứ ba, Cambridge, Vương quốc Anh, ngày 21-23 tháng 2 năm 1996, Kỷ yếu*, tập 1039 của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 83–87. Springer, 1996.

[42] Christos Patsonakis, Katerina Samari, Mema Roussopoulos và Aggelos Kiayias. Hướng tới cơ sở hạ tầng khóa công khai, phi tập trung, dựa trên hợp đồng thông minh. Trong *CANS*, 2017.

[43] Michael K. Reiter, Matthew K. Franklin, John B. Lacy và Rebecca N. Wright. Dịch vụ quản lý khóa ω. CCS '96.

[44] Leonid Reyzin và Sophia Yokoubov. Bộ tích lũy không đồng bộ hiệu quả cho PKI phân tán. Trong Vassilis Zalos và Roberto De Prisco, các biên tập viên, *Bảo mật và Mật mã cho Mạng - Hội nghị Quốc tế lần thứ 10, SCN 2016, Amalfi, Ý, ngày 31 tháng 8 - ngày 2 tháng 9 năm 2016, Kỷ yếu, tập 9841* của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 292–309. Springer, 2016.

[45] Barkley Rosser. Giới hạn rõ ràng cho một số hàm của số nguyên tố. *Tạp chí Toán học Hoa Kỳ*, 63:211–232, 1941.

[46] Tomas Sander. Bộ tích lũy hiệu quả không có bản tóm tắt mở rộng về cửa bí mật. Trong Vijay Varadharajan và Yi Mu, biên tập viên, *An ninh thông tin và truyền thông, Hội nghị quốc tế lần*

thứ hai, ICICS'99, Sydney, Úc, ngày 9-11 tháng 11 năm 1999, Kỷ yếu , tập 1726 của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 252–262. Springer, 1999.

[47] Tomas Sanderand Amnon Ta-Shma và Moti Yung. Bằng chứng thành viên mù, có thể kiểm tra được. Trong Yair Frankel, biên tập viên, *Mật mã tài chính, Hội nghị quốc tế lần thứ 4, FC 2000 Anguilla, Tây Ấn thuộc Anh, ngày 20-24 tháng 2 năm 2000, Kỷ yếu*, tập 1962 của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 53–71. Springer, 2000.

[48] Rita H. Wouhaybi và Andrew T. Campbell. Keypeer: Một hệ thống khóa công khai phân tán có khả năng mở rộng và linh hoạt sử dụng hợp âm, 2008.

[49] Zuoxia Yu, Man Ho Au, Rupeng Yang, Junzuo Lai và Qiuliang Xu. Bộ tích lũy phổ quát dựa trên mạng với các đối số không phải là thành viên. Trong Willy Susilo và Guomin Yang, biên tập viên, *Bảo mật và quyền riêng tư thông tin - Hội nghị Úc lần thứ 23, ACISP 2018, Wollongong, NSW, Úc, ngày 11-13 tháng 7 năm 2018, Kỷ yếu, tập* 10946 của *Ghi chú Bài giảng về Khoa học Máy tính*, trang 502–519. Springer, 2018.

[50] Emre Yuce và Ali Aydin Selcuk. Thủ quỹ máy chủ: Một cách tiếp cận bổ sung cho mô hình tin cậy PKI trên web. *Lưu trữ ePrint về Mật mã học IACR*, 2016:126, 2016.

[51] Lidong Chu, Fred B. Schneider và Robbert Van Renesse. Coca: Cơ quan chứng nhận trực tuyến được phân phối an toàn. *ACM Trans. Máy tính. Hệ thống. 2002*.

[52] Philip Zimmerman. Sự riêng tư khá tốt. https://philzimmermann.com.

Chứng minh định lý 6.1

Định lý 6.1. *Giao thức π của Hình 7 thực hiện một cách an toàn hàm $\mathcal{F}_{ns}$ trong thế giới hỗn hợp ($\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$) theo giả RSA mạnh trong mô hình Oracle ngẫu nhiên. Cụ thể, đối với bất kỳ kẻ tấn công ppt $\mathcal{A}$ nào tương tác với giao thức π trong thế giới hỗn hợp ($\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$), có một trình mô phỏng ppt $\mathcal{S}$ tương tác với hàm $\mathcal{F}_{ns}$ sao cho đối với mọi môi trường ppt $\mathcal{Z}$, nó cho thấy rằng*

$$EXEC^{\mathcal{F}_{ns}}_{\mathcal{Z},\mathcal{S}} \overset{c}{\approx} EXEC^{\mathcal{F}_{TP},\mathcal{F}_{UDB}}_{\mathcal{Z},\mathcal{A}}.$$

Chứng minh. Chúng tôi xây dựng một trình mô phỏng $\mathcal{S}$ (Hình 8) mô phỏng phép thực thi của giao thức π trong thế giới hỗn hợp ($\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$), với sự có mặt của kẻ tấn công $\mathcal{A}$. $\mathcal{S}$ đóng vai trò *T*, $\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$ đóng vai trò của máy chủ và hoạt động thay mặt cho một số máy trạm trung thực trong việc mô phỏng giao thức không gian hỗn hợp π. Dựa trên thiết kế trình mô phỏng $\mathcal{S}$, về cơ bản, chúng tôi chỉ ra rằng một môi trường có thể phân biệt giữa các phép thực thi trong thế giới hỗn hợp và thế giới lý tưởng *chỉ* bằng cách tác động đến cách diễn ra các thử nghiệm thành viên hoặc không phải thành viên trong giao thức thế giới hỗn hợp. Nói cách khác, sự mâu thuẫn duy nhất giữa hai phép thực thi có thể được rút ra nếu kẻ tấn công cố gắng thuyết phục hàm $\mathcal{F}_{TP}$ về các tuyên bố sai (về việc một phần tử có thuộc về một tập hợp tích lũy hay không) bằng cách phá vỡ tính bảo mật của ít nhất một trong các bộ tích lũy của giao thức π. Lưu ý rằng quan hệ *R*(*pk*, **aux**) không tạo cơ hội để phân biệt vì nó giống nhau ở cả hai thế giới.

Giả sử $\mathcal{Z}$ là môi trường ppt. Đối với bất kỳ thông điệp nào được $\mathcal{Z}$ gửi đến một bên (ví dụ: máy trạm *C* hoặc bên *T*), chúng tôi kiểm tra đầu ra trong cả thế giới hỗn hợp và thế giới lý tưởng. Lưu ý rằng trong mô hình UC, các bên trong thế giới lý tưởng là đơn giản, nghĩa là họ chỉ chuyển tiếp

bất kỳ thông điệp nào họ nhận được từ môi trường tới hàm và ngược lại. Dưới đây, chúng tôi chỉ ra rằng đối với bất kỳ thông báo nào được gửi bởi môi trường, đầu ra của các bên trong thế giới hỗn hợp và thế giới lý tưởng là không thể phân biệt được và do đó môi trường không thể phân biệt giữa phép thực thi trong thế giới hỗn hợp và thế giới lý tưởng.

$\mathcal{Z}$ gửi $(sid, \mathbf{Init})$ đến máy chủ S_i: Trong thế giới hỗn hợp, S_i gửi (*sid*, **InitTP**) đến $\mathcal{F}_{TP}$ và (*sid*, **InitUDB**) đến $\mathcal{F}_{UDB}$. Các hàm $\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$ thêm máy chủ S_i vào tập S_{init} và S'_{init} tương ứng và thông báo cho kẻ tấn công $\mathcal{A}$ rằng S_i được khởi tạo. Cả hai đều trả về **success** cho S_i, từ đó lần lượt trả về **success**. Trong thế giới lý tưởng, trình mô phỏng $\mathcal{S}$, khi nhận (*sid*, **Init**, S_i) từ $\mathcal{F}_{ns}$, theo Bước 1 của Hình 8, nó sẽ gửi **allow** tới $\mathcal{F}_{ns}$, trả về **success** cho S_i.

$\mathcal{Z}$ gửi $(sid, \mathbf{Setup}, R)$ cho bên T: Trong thế giới hỗn hợp, bên *T* gửi (*sid*, **Install**, *P*) tới $\mathcal{F}_{TP}$. Nếu **flag** = *ready* và $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{TP}$ lưu trữ *P*, thiết lập *state* ← ε và trả về **success** cho *T*. Sau đó, *T* chạy **KeyGen** (1^{λ}) hai lần (Bước 2, Hình 7), thiết lập *params* = (pk_1, pk_2, *R*) và tiếp theo, gửi (*sid*, (**Setup**, *params*)) tới $\mathcal{F}_{TP}$. Nếu $\mathcal{A}$ trả về **allow**, $\mathcal{F}_{TP}$ chạy *P* với đầu vào (**Setup**, *params*) và trả về *state* ← ($c_{0,1}$, $c_{0,2}$, *params*) cho *T* và *T* trả về **success** cho $\mathcal{Z}$. Trong thế giới lý tưởng, giả sử **flag** = **start**, hàm $\mathcal{F}_{ns}$ gửi (*sid*, **Setup**, *R*) đến $\mathcal{S}$. Trình mô phỏng $\mathcal{S}$ tuân theo Bước 2 của Hình 8, mô phỏng *T* và $\mathcal{F}_{TP}$ trong phép thực thi giao thức π với sự có mặt của $\mathcal{A}$. Cho rằng $\mathcal{A}$ trả về **allow** cho $\mathcal{S}$ khi đóng vai trò là $\mathcal{F}_{TP}$, $\mathcal{S}$ trả về **allow** cho $\mathcal{F}_{ns}$. Sau đó, $\mathcal{F}_{ns}$ trả về **success** cho *T*. Vì vậy, *T* trả về **success** cả ở thế giới hỗn hợp và thế giới lý tưởng.

$\mathcal{Z}$ gửi thông điệp $(sid, \mathbf{Register}, id, pk)$ cho máy trạm C: Chúng tôi sẽ phân biệt các trường hợp khác nhau liên quan đến việc kẻ tấn công $\mathcal{A}$ có gửi thông điệp đến $\mathcal{F}_{UDB}$ làm thay đổi nội dung của cơ sở dữ liệu trước khi $\mathcal{Z}$ gửi (*sid*, **Register**, *id*, *pk*) đến *C* hay không. Trong mọi trường hợp, chúng tôi sẽ kiểm tra cả đầu ra của máy trạm *C* trung thực và máy trạm *C* gian lận (tức là máy trạm bị kẻ tấn công $\mathcal{A}$ kiểm soát) cả trong thế giới hỗn hợp và thế giới lý tưởng. Chúng ta xét ba trường hợp sau:

Reg 1: $\mathcal{A}$ **chưa** gửi (*sid*, **ChangeDBstate**, *DBstate'*) đến $\mathcal{F}_{UDB}$ trước khi (*sid*, **Register**, *id*, *pk*) được gửi đến máy trạm *C*. Dưới đây, chúng tôi kiểm tra hai trường hợp nhỏ liên quan đến việc danh tính *id* đã được đăng ký hay chưa:

Reg 1(a): *Danh tính id chưa được đăng ký:* Trong thế giới hỗn hợp, một máy trạm *trung thực C* gửi (*sid*, **RetrieveDB**) đến $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì *C* nhận *DBstate* và tính toán bằng chứng W_1, W_2 theo Bước 3a, 3b của Hình 7. Sau đó, *C* gửi (*sid*, (**Register**, *id*, *pk*, *i* + 1, W_1, W_2)) đến $\mathcal{F}_{TP}$. Giả sử $\mathcal{A}$ trả về **allow** đến $\mathcal{F}_{TP}$, $\mathcal{F}_{TP}$ trả về ((c'_1, W'_1), (c'_2, W'_2), *state*) và *C* tính toàn bằng chứng W'_3. Sau đó, *C* gửi (*sid*, **Post,** (**Register**, *id*, *pk*, *i* + 1, W'_1, W'_2, W'_3) đến $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì *C* trả về **success** trong thế giới hỗn hợp. Trong thế giới lý tưởng, máy trạm *C* cũng trả về **success**, bởi vì trình mô phỏng $\mathcal{S}$, như có thể thấy trong Bước 3 của Hình 8, trả về **allow** cho $\mathcal{F}_{ns}$, đóng vai trò *C*, $\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$. Cuối cùng, $\mathcal{F}_{ns}$ xác minh *id* chưa được đăng ký, sẽ gửi **success** tới *C*.

Trình mô phỏng $\mathcal{S}$:

1. Khi nhận được (*sid*, **Init**, S_i) bởi $\mathcal{F}_{ns}$, $\mathcal{S}$ thay mặt $\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$ gửi (*sid*, **InitTP**, S_i) và (*sid*, **InitUDB**, S_i) đến $\mathcal{A}$ và gửi **allow** đến $\mathcal{F}_{ns}$.

2. Khi nhận được (*sid*, **Setup**, *R*) bởi $\mathcal{F}_{ns}$, $\mathcal{S}$ thay mặt cho bên *T* trong giao thức thế giới thực, chạy **KeyGen** (1^{λ}) hai lần (theo Bước 2 của Hình 7) và sau đó, đóng vai trò của hàm $\mathcal{F}_{TP}$ trong giao thức thế giới thực, gửi (*sid*, **Install**, *P*) tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ sẽ gửi (*sid*, **Setup**, *params*) đến $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ chạy *P* với đầu vào (**Setup**, *params*) và gửi **allow** tới $\mathcal{F}_{ns}$.

3. Khi nhận được (*sid*, **Register**, *id*, *pk*) bởi $\mathcal{F}_{ns}$, $\mathcal{S}$ đóng vai trò là một máy trạm trung thực *C* và vai trò là $\mathcal{F}_{UDB}$ trong giao thức thế giới lai π, gửi (*sid*, **RetrieveDB**, *C*) đến $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ sẽ chạy Bước 3a của Hình 7. Nếu bản ghi cuối cùng liên quan đến *id* là bản ghi đăng ký, $\mathcal{S}$ sẽ gửi **fail** tới $\mathcal{F}_{ns}$, nếu không thì nó sẽ tiếp tục chạy Bước 3b nếu cần. Sau đó, $\mathcal{S}$, đóng vai trò là $\mathcal{F}_{TP}$ trong giao thức thế giới thực, gửi (*sid*, (**Register**, *id*, *pk*, *i* + 1, W_1', W_2') tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ chạy *P* với đầu vào (**Register**, *id*, *pk*, *i* + 1, W_1', W_2'). Nếu P xuất ra **fail** thì $\mathcal{S}$ gửi **fail** tới $\mathcal{F}_{ns}$, nếu không thì *P* trả về $((c_1', W_1'), (c_2', W_2'), state)$ và $\mathcal{S}$ tính toán một bằng chứng không phải là thành viên W_3' cho (*id*, *i* + 1, *d*). Sau đó, thay mặt $\mathcal{F}_{UDB}$, $\mathcal{S}$ gửi (*sid*, **Post**, (**Register**, *id*, *pk*, *i*, W_1', W_2', W_3') tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ gửi **allow** tới $\mathcal{F}_{ns}$ và cập nhật *DBstate* bằng cách lưu trữ (**Register**, *id*, *pk*, *i*, W_1', W_2', W_3'), như $\mathcal{F}_{UDB}$ thực hiện trong Hình 6.

4. Khi nhận được (*sid*, **Revoke**, *id*, *pk*, **aux**) bởi $\mathcal{F}_{ns}$, $\mathcal{S}$ đóng vai trò một máy trạm trung thực *C* và vai trò là $\mathcal{F}_{UDB}$, gửi (*sid*, **RetrieveDB**, *C*) đến $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ chạy Bước 4a, 4b của Hình 7 và tính toán các bằng chứng được cập nhật W_1', W_2', W_3'. Sau đó, $\mathcal{S}$ trên thay mặt cho $\mathcal{F}_{TP}$, gửi (*sid*, **Revoke**, *id*, *pk*, *i*, W_1', W_2', W_3', **aux**) tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ chạy *P* với đầu vào (**Revoke**, *id*, *pk*, *i*, W_1', W_2', W_3', **aux**) và nếu nó xuất ra (**fail**, *state*) thì $\mathcal{S}$ gửi **fail** tới $\mathcal{F}_{ns}$. Ngược lại, $\mathcal{S}$ trả về **allow** cho $\mathcal{F}_{ns}$ và cập nhật *DBstate* bằng cách lưu trữ (**Revoke**, *id*, *pk*, *i*).

5. Khi nhận được (*sid*, **Retrieve**, *id*) bởi $\mathcal{F}_{ns}$, $\mathcal{S}$ thay mặt cho một máy trạm trung thực *C* và đóng vai trò là $\mathcal{F}_{UDB}$, gửi (*sid*, **RetrieveDB**) đến $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{S}$ sẽ chạy Bước 5a của Hình 7. Nếu Bước 5a trả về **fail** thì $\mathcal{S}$ trả về **fail** cho $\mathcal{F}_{ns}$, nếu không thì $\mathcal{S}$ chạy Bước 5b và mô phỏng $\mathcal{F}_{TP}$, gửi (*sid*, **RetrieveState**) tới $\mathcal{A}$. Nếu $\mathcal{A}$ trả về **allow** và tất cả các thuật toán ở Bước 5b trả về 1 thì $\mathcal{S}$ gửi **allow** tới $\mathcal{F}_{ns}$, nếu không thì gửi **fail**.

6. Khi nhận được (*sid*, **VerifyID**, *id*) hoặc (*sid*, **VerifyMapping**, *id*, *pk*) bởi $\mathcal{F}_{ns}$, $\mathcal{S}$ chạy mô phỏng tương tự như Bước 5.

7. Khi nhận được (*sid*, **ChangeDBstate**, *DBstate'*) bởi $\mathcal{A}$, $\mathcal{S}$ thiết lập *DBstate* ← *DBstate'*.

8. Khi nhận được (*sid*, **Register**, *id*, *pk*) hoặc (*sid*, **Revoke**, *id*, *pk*) bởi $\mathcal{F}_{ns}$ đối với máy trạm *C gian lận*, $\mathcal{S}$ sẽ chờ đợi hành động của $\mathcal{A}$ và mô phỏng $\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$ như trong các trường hợp trước. Nếu $\mathcal{S}$ nhận được (**Register**, *id'*, *pk'*, *i*, W_1', W_2'), kiểm tra xem $id \neq id'$ hoặc/và $pk \neq pk'$. Nếu chương trình *P* không trả về (**fail**, *state*) với đầu vào này thì $\mathcal{S}$ sẽ gửi (**Register**, *id'*, *pk'*, *C*) đến $\mathcal{F}_{ns}$. $\mathcal{S}$ chạy tương tự khi nhận được (**Revoke**, *id'*, *pk'*, *i*, W_1', W_2', W_3').

Hình 8: Trình mô phỏng $\mathcal{S}$.

Reg 1(b): *Danh tính id hiện đã được đăng ký:* Trong thế giới hỗn hợp, một *C trung thực* sẽ gửi (*sid*, **RetrieveDB**) đến $\mathcal{F}_{UDB}$ và khi *C* nhận được *Dbstate*, kiểm tra *id* đã được đăng ký và trả về **fail**. Trong thế giới lý tưởng, $\mathcal{S}$ mô phỏng *C* và $\mathcal{F}_{UDB}$ theo Bước 3 của Hình 8, kiểm tra xem có bản ghi đăng ký cho danh tính *id* chưa bị thu hồi không và do đó trả về **fail** cho $\mathcal{F}_{ns}$. Tiếp theo, $\mathcal{F}_{ns}$ trả về **fail** cho *C*.

Một máy trạm *C gian lận* trong thế giới hỗn hợp có thể cố gắng thuyết phục $\mathcal{F}_{TP}$ rằng *id* **chưa** được đăng ký. Sau đó, *C* nên cung cấp một bằng chứng không phải là thành viên W_1 cho (*id*, *j*, *a*) với $j \geq 2$, sao cho **VerifyNonMem**(pk_2, (*id*, *j*, *a*), W_1, c_2) = 1 và một bằng chứng thành viên W_2 cho (*id*, $j-1$, *d*) sao cho **VerifyMem**(pk_2, (*id*, $j-1$, *d*), W_2, c_2) = 1, hoặc *C* nên cung cấp một bằng chứng không phải thành viên hợp lệ W_1 cho (*id*, 1, *a*). Chúng tôi thấy rằng nhờ tính bảo mật của bộ tích lũy c_2, một cuộc tấn công như vậy chỉ diễn ra với xác suất không đáng kể. Hãy nhớ lại rằng vì *id* hiện đã được đăng ký nên (1) có $\ell \geq 2$ sao cho (*id*, ℓ, *a*) $\in X_2$ và (*id*, ℓ, *d*) $\notin X_2$, trong đó X_2 được thiết lập tích lũy trong c_2, hoặc, (2) (*id*, 1, *a*) $\in X_2$. Bắt đầu với (1), chúng tôi xem xét các trường hợp $1 < j \leq \ell$ *và* $j > \ell$. Nếu $1 < j \leq \ell$ thì (*id*, *j*, *a*) $\in X_2$ và (*id*, $j-1$, *d*) $\in X_2$. Bằng tính bảo mật của bộ tích lũy c_2, *C* có thể tạo ra một bằng chứng không phải là thành viên hợp lệ W_1 cho (*id*, *j*, *a*) chỉ với xác suất không đáng kể. Nếu $j > \ell$ thì (*id*, *j*, *a*) $\notin X_2$ và (*id*, $j-1$, *d*) $\notin X_2$. Do tính bảo mật của bộ tích lũy c_2, *C* không thể tạo ra bằng chứng thành viên hợp lệ W_2 cho (*id*, $j-1$, *d*). Đối với trường hợp (2), tương tự, *C* chỉ có thể đưa ra một bằng chứng không phải là thành viên hợp lệ W_1 cho (*id*, 1, *a*) với xác suất không đáng kể. Do đó, nếu một máy trạm *C gian lận* gửi (*sid*, **Register**, *id*, *pk*, *j*, W_1, W_2) đến $\mathcal{F}_{TP}$, theo lý do được mô tả ở trên, $\mathcal{F}_{TP}$ sẽ trả về (**fail**, *state*) (ngoại trừ với xác suất không đáng kể). Do đó, *C* trả về **fail** trong thế giới hỗn hợp. Kiên định với thế giới hỗn hợp, trong thế giới lý tưởng, *C* cũng sẽ trả về **fail**. Cụ thể, trình mô phỏng $\mathcal{S}$, theo Bước 8, chờ đợi hành động của $\mathcal{A}$, tức là máy trạm *C* gian lận. Sau đó, $\mathcal{S}$ mô phỏng $\mathcal{F}_{TP}$ trong tầm kiểm soát của máy trạm *C* gian lận và vì $\mathcal{F}_{TP}$ trả về (**fail**, *state*), $\mathcal{S}$ gửi **fail** tới $\mathcal{F}_{ns}$. Sau đó, $\mathcal{F}_{ns}$ gửi **fail** tới *C* và *C* trả về **fail**.

Reg 2: $\mathcal{A}$ đã gửi (*sid*, **ChangeDBstate**, *DBstate*′) đến $\mathcal{F}_{UDB}$ trước khi $\mathcal{Z}$ gửi (*sid*, **Register**, *id*, *pk*), sao cho *DBstate*′ ≠ *DBstate* và tập hợp X_2' lấy từ *DBstate*′ khác với tập hợp X_2 tích lũy trong c_2[4]. Chúng tôi xem xét các trường hợp nhỏ sau:

Reg 2(a): *Danh tính id chưa được đăng ký nhưng bản ghi cuối cùng bao gồm id trong DBstate*′ *có dạng* (**Register**, *id*, *pk*, *j*, W_1, W_2, W_3)*:* Trong thế giới hỗn hợp, một *C trung thực* gửi (*sid*, **RetrieveDB**) đến $\mathcal{F}_{UDB}$ và khi *C* nhận được *DBstate*′, kiểm tra xem *id* đã được đăng ký chưa và trả về **fail**. Trong thế giới lý tưởng, $\mathcal{S}$ mô phỏng *C* và $\mathcal{F}_{UDB}$, gửi **fail** tới $\mathcal{F}_{ns}$, trả về **fail** cho *C*. Một *C gian lận* có thể gửi (**Register**, *id*, *pk*′, *j*′, W_1', W_2') đến $\mathcal{F}_{TP}$. Trong trường hợp đó, nếu $\mathcal{F}_{TP}$ trả về (**fail**, *state*) thì *C* sẽ trả về **fail**. Trong thế giới lý tưởng, $\mathcal{S}$ mô phỏng $\mathcal{F}_{TP}$, trả về **fail** cho $\mathcal{F}_{ns}$, trả về **fail** cho *C*. Nếu $\mathcal{F}_{ns}$ trả về ((c_1', W_1'), (c_2', W_2'), *state*) và $\mathcal{F}_{UDB}$ trả về **success** cho *C* sau khi nhận được thông điệp có dạng (*sid*, **Post**,·) thì *C* xuất ra **success**. Tương ứng, trong thế giới lý tưởng $\mathcal{S}$, mô phỏng

[4]Như chúng tôi đã giải thích trong Phần 5, một tập hợp X_2' lấy từ *DBstate*′ theo cách sau: Với mọi bản ghi có dạng (**Register**, *id*, *pk*, *i*, W_1, W_2, W_3), (*id*, *i*, *a*) được thêm vào X_2 và đối với bất kỳ bản ghi nào có dạng (**Revoke**, *id*, *pk*, *i*), (*id*, *i*, *d*) được thêm vào X_2.

$\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$, trả về **allow** cho $\mathcal{F}_{ns}$, đầu tiên kiểm tra xem có phải *id* chưa được đăng ký, thêm cặp (id, pk') và gửi **success** tới *C*. Trong cả hai trường hợp, *C* trả về kết quả đầu ra nhất quán trong thế giới lý tưởng và thế giới hỗn hợp.

Reg 2(b): *Danh tính id chưa được đăng ký và bản ghi cuối cùng bao gồm id trong DBstate′ có dạng* (**Revoke**, *id*, *pk*, *j*) *hoặc không có bản ghi nào cho id:* Lý do trong trường hợp nhỏ này là tương tự với Reg2(a), ngoại trừ việc một máy trạm trung thực tương tác với $\mathcal{F}_{TP}$ sau khi nhận được *DBstate′* từ $\mathcal{F}_{UDB}$.

Reg 2(c): *Danh tính id được đăng ký và bản ghi cuối cùng bao gồm id trong DBstate′ có dạng* (**Revoke**, *id*, *pk*, *j*) *hoặc không có bản ghi nào cho id:* Trong thế giới hỗn hợp, một máy trạm *trung thực C* gửi (*sid*, **RetrieveDB**) tới $\mathcal{F}_{UDB}$. Khi nhận *DBstate′* từ $\mathcal{F}_{UDB}$, nó chạy Bước 3a, 3b của giao thức π (Hình 7) và tính toán bằng chứng không phải là thành viên W_1 cho (*id*, *j* + 1, *a*) và bằng chứng tư cách thành viên W_2 cho (*id*, *j*, *d*) hoặc thiết lập $W_2 = \perp$ trong trường hợp không có bản ghi nào cho *id* trong *DBstate′*. Sau đó, *C* gửi (*sid*, **Register**, *id*, *pk*, *j* + 1, W_1, W_2) đến $\mathcal{F}_{TP}$. Giả sử một máy trạm trung thực *C* với tập hợp tích lũy $X_2' \neq X_2$ có thể tạo ra một bằng chứng không phải là thành viên hợp lệ W_1 cho (*id*, *j* + 1, *a*) và một bằng chứng thành viên hợp lệ W_2 cho (*id*, *j*, *d*) với xác suất không đáng kể theo các thuật toán tạo thành viên và không phải là thành viên tương ứng thì tính bảo mật của bộ tích lũy c_2 sẽ bị phá vỡ, sử dụng các đối số tương tự như trong trường hợp Reg1(b). Do đó, $\mathcal{F}_{TP}$ trả về (**fail**, *state*) cho *C* và *C* trả về **fail** trong thế giới hỗn hợp. Máy trạm *C* cũng trả về **fail** trong thế giới lý tưởng, vì $\mathcal{S}$ mô phỏng $\mathcal{F}_{TP}$ và *C* và trả về **fail** cho $\mathcal{F}_{ns}$.

Một *C gian lận* trong thế giới hỗn hợp có thể gửi (**Register**, *id*, pk^*, ℓ, W_1^*, W_2^*) để thuyết phục $\mathcal{F}_{TP}$ rằng *id* chưa được đăng ký. Theo phân tích tương tự như trong trường hợp Reg1(b), chúng tôi có thể kết luận rằng *C* trả về cùng một đầu ra cả trong thế giới hỗn hợp và thế giới lý tưởng.

Reg 2(d): *Danh tính id đã được đăng ký nhưng bản ghi cuối cùng bao gồm id trong DBstate′ có dạng* (**Register**, *id*, *pk*, *i* +1, W_1, W_2, W_3)*:* Theo các đối số của Reg2(c) với điểm khác biệt duy nhất là một máy trạm trung thực trả về **fail** sau khi nhận được *DBstate′* từ $\mathcal{F}_{UDB}$, chúng tôi kết luận rằng *C* trả về các kết quả đầu ra nhất quán trong thế giới lý tưởng và thế giới hỗn hợp.

Reg 3: *$\mathcal{A}$ đã gửi* (*sid*, **ChangeDBstate**, *DBstate′*) *đến $\mathcal{F}_{UDB}$ trước* (*sid*, **Register**, *id*, *pk*) *do $\mathcal{Z}$ gửi, sao cho DBstate′ ≠ DBstate nhưng tập hợp X_2' lấy từ DBstate′ giống với X_2 tích lũy trong c_2.* Trong trường hợp này, có thể áp dụng lý do tương tự với Reg1, trong đó kẻ tấn công chưa gửi thôn điệp làm thay đổi nội dung được lưu trữ bởi $\mathcal{F}_{UDB}$. Điều này xảy ra vì tập hợp tích lũy vẫn giữ nguyên và do đó một máy trạm trung thực có thể tính toán chính xác các bằng chứng W_1, W_2.

$\mathcal{Z}$ gửi $(sid, \textbf{Revoke}, id, pk, \textbf{aux})$ cho máy trạm C: Tương tự như trường hợp trước, chúng tôi kiểm tra đầu ra của máy trạm *C* trong thế giới hỗn hợp và thế giới lý tưởng bằng cách phân biệt các trường hợp khác nhau tùy thuộc vào việc liệu kẻ tấn công có giả mạo nội dung được lưu trữ bởi $\mathcal{F}_{UDB}$ hay không. Lưu ý rằng trong tất cả các trường hợp mô tả bên dưới, chúng tôi giả định

rằng $R(pk, \textbf{aux}) = 1$, nếu không, có thể dễ dàng nhận thấy rằng máy trạm C trả về **fail** trong cả trong thế giới lý tưởng và thế giới hỗn hợp.

Rev 1: $\mathcal{A}$ **chưa** gửi (sid, **ChangeDBstate**, $DBstate'$) đến $\mathcal{F}_{UDB}$ trước khi (sid, **Revoke**, id, pk, **aux**) được gửi đến C. Chúng tôi xem xét hai trường hợp nhỏ khác nhau:

Rev 1(a): (id, pk) *đã được đăng ký:* Trong thế giới hỗn hợp, một máy trạm C *trung thực* gửi (sid, **RetrieveDB**) tới $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ sẽ gửi $DBstate$ tới C. C tính toán các bằng chứng W_1', W_2', W_3' theo các bước 4a, 4b của giao thức π trong Hình 7. Hãy nhớ lại rằng W_1' là bằng chứng thành viên cho (id, i, a), W_2' là bằng chứng không phải thành viên cho (id, i, d) và W_3' là bằng chứng thành viên cho (id, pk, i, a). Sau đó, C gửi (sid, **Revoke**, id, pk, i, W_1', W_2', W_3', **aux**) đến $\mathcal{F}_{TP}$. Nếu $\mathcal{A}$ trả về **allow** và vì $R(pk, \textbf{aux}) = 1$, nên $\mathcal{F}_{TP}$ trả về ((c_1', W_1'), (c_2', W_2'), $state$). Sau đó C gửi (sid, **Post**, (**Revoke**, id, pk, i)) tới $\mathcal{F}_{UDB}$ và nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ trả về **success** cho C. Trong thế giới lý tưởng, $\mathcal{S}$, khi nhận được (sid, **Revoke**, id, pk, **aux**) mô phỏng C, $\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$ như được mô tả trong Bước 4 của Hình 8. $\mathcal{A}$ trả về **allow** dưới dạng phản hồi trong mọi trường hợp nó được yêu cầu, $\mathcal{S}$ gửi **allow** tới $\mathcal{F}_{ns}$. Sau đó, $\mathcal{F}_{ns}$ kiểm tra xem $R(pk, \textbf{aux}) = 1$ và (id, pk) $\in X$, nó xóa cặp (id, pk) và gửi **success** đến C. Do đó, một máy trạm trung thực C trả về **success** cả trong thế giới hỗn hợp và thế giới lý tưởng.

Rev 1(b): (id, pk) *chưa được đăng ký:* Trong thế giới hỗn hợp, một máy trạm C *trung thực* gửi (sid, **RetrieveDB**) tới $\mathcal{F}_{UDB}$ và nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ trả về $DBstate$ cho C. C kiểm tra xem bản ghi cuối cùng liên quan đến id có phải là bản ghi đăng ký hay không và vì bản ghi này không chứa được nên C trả về **fail**. Trong thế giới lý tưởng, $\mathcal{S}$, mô phỏng C và $\mathcal{F}_{UDB}$, gửi **fail** tới $\mathcal{F}_{ns}$, sau đó gửi **fail** tới C. Kết quả là C cũng trả về **fail** trong thế giới lý tưởng.

Một C *gian lận* trong thế giới hỗn hợp có thể cố gắng thuyết phục $\mathcal{F}_{TP}$ rằng (id, pk) *hiện tại* đã được đăng ký. Chúng tôi sẽ xem xét ba trường hợp. Đầu tiên, chúng tôi sẽ xem xét trường hợp danh tính id chưa bao giờ được đăng ký, thứ hai là trường hợp id hiện chưa được đăng ký nhưng id đã được đăng ký trong quá khứ, và thứ ba là trường hợp một cặp (id, pk') hiện đã đăng ký, trong đó $pk' \neq pk$.

- Trong trường hợp đầu tiên, khi danh tính id chưa bao giờ được đăng ký, C nên tìm i và tính toán bằng chứng thành viên W_1' cho (id, i, a) và bằng chứng không phải thành viên W_2' cho (id, i, d). Tuy nhiên, vì id chưa bao giờ được đăng ký nên (id, i, a) $\notin X_2$ và (id, i, d) $\notin X_2$. Bằng tính bảo mật của bộ tích lũy c_2, kẻ tấn công chỉ có thể tìm thấy bằng chứng thành viên W_1' với xác suất không đáng kể. Do đó, $\mathcal{F}_{TP}$ sẽ trả về **fail** và C trả về **fail** trong thế giới hỗn hợp ngoại trừ xác suất không đáng kể.
- Chúng ta tiếp tục với trường hợp thứ hai, trong đó id đã được đăng ký trước đây. Tương tự như trường hợp đầu tiên, C nên tìm i và tính toán bằng chứng tư cách thành viên W_1' cho (id, i, a) và bằng chứng không phải thành viên W_2' cho (id, i, d). Giả sử id đã được đăng ký trong quá khứ ℓ lần và C chọn $i \in \{1,\ldots,\ell\}$. Sau đó, C phải tính toán bằng chứng tư cách thành viên W_1' cho (id, i, a) và bằng chứng không phải thành viên W_2' cho (id, i, d). Tuy nhiên, chúng ta có (id, i, a) $\in X_2$ và (id, i, d) $\in X_2$ vì id hiện chưa được đăng ký. Bằng tính bảo mật của bộ tích lũy c_2, C có thể tính toán một bằng chứng không phải là thành viên cho (id, i, d) $\in X_2$ chỉ với xác suất không đáng kể và

do đó C trả về **fail**. Nếu C chọn $i > \ell$, theo các đối số tương tự với trường hợp đầu tiên, C trả về **fail** ngoại trừ với xác suất không đáng kể (do tính bảo mật của bộ tích lũy c_2).

- Trong trường hợp thứ ba, vì (id, pk') đã được đăng ký, nên có ℓ sao cho $(id, \ell, a) \in X_2$, $(id, \ell, d) \notin X_2$, $(id, pk', \ell, a) \in X_1$ và $(id, pk', i, d) \notin X_1$. Giả sử C chọn ℓ thì C phải tính toán bằng chứng tư cách thành viên W_1' cho $(id, \ell, a) \in X_1$, bằng chứng không phải thành viên W_2' cho $(id, \ell, d) \notin X_2$ và bằng chứng tư cách thành viên W_3' cho $(id, pk, \ell, a) \in X_1$[5]. Do tính bảo mật của bộ tích lũy c_1, C không thể tính toán bằng chứng thành viên hợp lệ cho $(id, pk, \ell, a) \notin X_1$. Do đó, C trả về **fail**, ngoại trừ xác suất không đáng kể. Nếu C chọn $i < \ell$ hoặc $i > \ell$, theo các lập luận tương tự với trường hợp thứ nhất và thứ hai, nhờ tính bảo mật của bộ tích lũy c_2, chúng tôi kết luận rằng C trả về **fail** ngoại trừ xác suất không đáng kể. Trong thế giới lý tưởng, trong tất cả các trường hợp đã nói ở trên, $\mathcal{S}$, mô phỏng $\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$ và vì $\mathcal{F}_{TP}$ trả về (**fail**, *state*), $\mathcal{S}$ gửi **fail** tới $\mathcal{F}_{ns}$, do đó, gửi **fail** tới C.

Rev 2: $\mathcal{A}$ đã gửi (*sid*, **ChangeDBstate**, *DBstate'*) đến $\mathcal{F}_{UDB}$ trước khi $\mathcal{Z}$ gửi (*sid*, **Revoke**, *id*, *pk*, **aux**), sao cho $DBstate' \neq DBstate$ ít nhất một trong các tập tích lũy X_i' ($i \in \{1, 2\}$) lấy từ $DBstate'$ khác với giá trị tương ứng được lấy bởi *DBstate*: Xét cách X_1', X_2' được tính toán cho $DBstate'$ (xem Phần 5), có hai trường hợp có thể xảy ra là (1) $X_2' \neq X_2$ và $X_1' \neq X_1$ và (2) $X_2' = X_2$ và $X_1' = X_1$. Đối với cả hai kịch bản, chúng tôi có các trường hợp nhỏ sau:

Rev 2(a): (*id*, *pk*) *đã được đăng ký*: Trong thế giới hỗn hợp, một *C trung thực* gửi (*sid*, **RetrieveDB**) đến $\mathcal{F}_{UDB}$ và nếu $\mathcal{A}$ trả về **allow** thì C sẽ tính toán các bằng chứng W_1', W_2', W_3' theo Bước 4a, 4b của Hình 7, và gửi (**Revoke**, *id*, *pk*, *j*, W_1', W_2', W_3') đến $\mathcal{F}_{TP}$. Nếu $\mathcal{F}_{TP}$ trả về **fail** thì C trả về **fail**, ngược lại, nếu $\mathcal{A}$ trả về **allow** sau khi C gửi (*sid*, **Post**, (**Revoke**, *id*, *pk*, *j*)) tới $\mathcal{F}_{UDB}$, C trả về **success** trong thế giới hỗn hợp. Trong thế giới lý tưởng, $\mathcal{S}$, mô phỏng C, $\mathcal{F}_{TP}$, $\mathcal{F}_{UDB}$, nếu $\mathcal{F}_{TP}$ trả về **fail** thì $\mathcal{S}$ trả về **fail** cho $\mathcal{F}_{ns}$ và $\mathcal{F}_{ns}$ trả về **fail** cho C. Ngay cả trong trường hợp $\mathcal{F}_{TP}$ trả về **success**, nghĩa là $\mathcal{S}$, mô phỏng $\mathcal{F}_{TP}$, sẽ trả về **allow** cho $\mathcal{F}_{ns}$. Vì (*id*, *pk*) đã được đăng ký, $\mathcal{F}_{ns}$ sẽ xóa cặp (*id*, *pk*) và nó sẽ trả về **success** cho máy trạm C. Do đó, máy trạm C trả về kết quả đầu ra nhất quán cả trong thế giới hỗn hợp hợp và thế giới lý tưởng.

Rev 2(b): (*id*, *pk*) *chưa được đăng ký*: Trong thế giới hỗn hợp hợp, một *C trung thực* gửi (*sid*, **RetrieveDB**) tới $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ trả về $DBstate'$ và sau đó C kiểm tra xem bản ghi đăng ký[6] của nó có tồn tại trong $DBstate'$ hay không. Nếu không tồn tại bản ghi như vậy thì C trả về **fail**. Nếu thế giới lý tưởng, $\mathcal{S}$ mô phỏng C và $\mathcal{F}_{TP}$, do đó $\mathcal{S}$ trả về **fail** cho $\mathcal{F}_{ns}$ và C cũng trả về **fail** trong thế giới lý tưởng. Nếu tồn tại một bản ghi như vậy thì C thực hiện theo Bước 4a, 4b của Hình 7, tính toán các bằng chứng W_1', W_2', W_3' và gửi (**Revoke**, *id*, *pk*, *j*, W_1', W_2', W_3') đến $\mathcal{F}_{TP}$. Bằng tính bảo mật của bộ tích lũy c_1, c_2, $\mathcal{F}_{TP}$ trả về (**fail**, *state*) ngoại trừ với xác suất không đáng kể. Chi tiết hơn, nếu một máy trạm trung thực được cung cấp $DBstate'$ có thể thuyết phục $\mathcal{F}_{TP}$ rằng (*id*, *pk*) hiện đã được đăng ký, sau đó tuân theo các lập luận tương tự như trong trường hợp

[5]Lưu ý rằng ngay cả khi (*id*, *pk*) đã được đăng ký trước đây, X_1 sẽ chứa một phần tử (*id*, *pk*, *j*, *a*) nhưng với $j \neq \ell$.

[6]Lưu ý rằng trong giao thức π của chúng tôi, máy trạm có trạng thái và do đó chúng lưu trữ bản ghi đăng ký cuối cùng.

Rev1(b), tính bảo mật của bộ tích lũy sẽ bị phá vỡ. Do đó, *C* trả về **fail** trong thế giới hỗn hợp. Trong thế giới lý tưởng, vì $\mathcal{S}$ mô phỏng *C*, $\mathcal{F}_{TP}$ gửi **fail** tới $\mathcal{F}_{ns}$ và trả về **fail** cho *C*. Do đó, *C* cũng trả về **fail** trong thế giới lý tưởng. Một máy trạm *C gian lận* có thể cố gắng thuyết phục $\mathcal{F}_{TP}$ rằng (*id*, *pk*) hiện đã được đăng ký (bất kể $\mathcal{F}_{UDB}$ đã gửi gì), bằng cách gửi (**Revoke**, *id*, *pk*, j^*, W_1^*, W_2^*, W_3^*) đến $\mathcal{F}_{TP}$. Bằng tính bảo mật của các bộ tích lũy c_1, c_2, tuân theo các đối số tương tự như trong Rev1(b), điều này chỉ xảy ra với xác suất không đáng kể. Do đó, *C* trả về **fail** trong cả trong thế giới hỗn hợp và thế giới lý tưởng.

Rev 3: $\mathcal{A}$ đã gửi (*sid*, **ChangeDBstate**, *DBstate*′) cho đến khi (*sid*, **Revoke**, *id*, *pk*, **aux**) được $\mathcal{Z}$ gửi, sao cho *DBstate*′ ≠ *DBstate* nhưng các tập hợp X_1', X_2' lấy từ *DBstate*′ giống với X_1, X_2: Trong trường hợp này, chúng tôi thực hiện phân tích tương tự với Rev1, vì một máy trạm trung thực có thể tính toán chính xác các bằng chứng W_1', W_2', W_3'.

$\mathcal{Z}$ gửi $(sid, \textbf{Retrieve}, id)$ cho máy trạm C: Chúng tôi xem xét các trường hợp sau:

Ret 1: $\mathcal{A}$ chưa gửi (*sid*, **ChangeDBstate**, *DBstate*′) đến $\mathcal{F}_{UDB}$ trước khi $\mathcal{Z}$ gửi (*sid*, **Retrieve**, *id*) tới *C*. Chúng tôi xem xét các trường hợp nhỏ sau:

Ret 1(a): (*id*, *pk*) *hiện đã được đăng ký*: Trong thế giới hỗn hợp, một máy trạm *trung thực C* gửi (*sid*, **RetrieveDB**) đến $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ gửi *DBstate* tới *C*, *C* thực hiện theo Bước 5a của giao thức π trong Hình 7 và tính toán các bằng chứng W_1', W_2', W_3'. Sau đó, *C* gửi (*sid*, **Retrievestate**) tới $\mathcal{F}_{TP}$. Nếu $\mathcal{A}$ trả về **allow** thì *C* sẽ chạy Bước 5b và trả về *pk* dưới dạng khóa công khai được truy xuất. Trong thế giới lý tưởng, vì $\mathcal{S}$ mô phỏng *C*, $\mathcal{F}_{TP}$ và $\mathcal{F}_{UDB}$, nên $\mathcal{S}$ trả về **allow** cho $\mathcal{F}_{ns}$. Sau đó $\mathcal{F}_{ns}$ kiểm tra xem có khóa công khai nào được đăng ký dưới danh tính *id* hay không và vì (*id*, *pk*) hiện đã được đăng ký nên $\mathcal{F}_{ns}$ trả về *pk* cho *C*. Do đó, *C* trả về kết quả đầu ra nhất quán cả trong thế giới hỗn hợp và thế giới lý tưởng.

Ret 1(b): *id hiện chưa được đăng ký:* Trong thế giới hỗn hợp, một máy trạm *trung thực C* gửi (*sid*, **RetrieveDB**) đến $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ sẽ gửi *DBstate* đến *C*. Sau đó *C* kiểm tra xem bản ghi cuối cùng liên quan đến *id* có phải là bản ghi đăng ký hay không. Vì điều này không đúng, *C* trả về **fail** trong thế giới hỗn hợp. Trong thế giới lý tưởng, $\mathcal{S}$, theo Bước 5 của Hình 8, sẽ trả về **fail** cho $\mathcal{F}_{ns}$ và sẽ trả về **fail** cho *C*. Do đó, *C* cũng trả về **fail** trong thế giới lý tưởng.

Một máy trạm *C gian lận* vẫn có thể gửi (*sid*, **Retrievestate**) đến $\mathcal{F}_{TP}$, tuy nhiên, *C* không có lợi thế trong việc ảnh hưởng đến đầu ra của $\mathcal{F}_{TP}$, do đó *C* trả về ⊥ cả trong thế giới hỗn hợp và thế giới lý tưởng.

Ret 2: $\mathcal{A}$ đã gửi (*sid*, **ChangeDBstate**, *DBstate*′) đến $\mathcal{F}_{UDB}$ trước khi $\mathcal{Z}$ gửi (*sid*, **Retrieve**, *id*) đến *C* sao cho tập tích lũy X_2' lấy từ *DBstate*′ khác với X_2. Chúng tôi xem xét các trường hợp nhỏ sau:

Ret 2(a): (*id*, *pk*) *hiện đã được đăng ký và bản ghi cuối cùng liên quan đến id là bản ghi đăng ký:* Trong thế giới hỗn hợp, một máy trạm *trung thực C* gửi (*sid*, **RetrieveDB**) đến $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ sẽ gửi *DBstate* tới *C*, *C* thực hiện theo Bước 5a của giao thức π trong Hình 7, tính toán các bằng chứng W_1', W_2', W_3'. Sau đó, *C* gửi (*sid*,

Retrievestate) tới $\mathcal{F}_{TP}$. Nếu $\mathcal{A}$ trả về **allow** thì *C* sẽ chạy Bước 5b của giao thức π. Ngay cả khi tất cả các thuật toán trả về 1 và *C* trả về *pk* trong thế giới hỗn hợp thì $\mathcal{S}$ sẽ trả về **allow** cho $\mathcal{F}_{ns}$ và vì (*id*, *pk*) được đăng ký nên *C* sẽ trả về *pk* trong thế giới lý tưởng. Nếu ít nhất một thuật toán trả về 0 thì *C* trả về $\perp$ trong thế giới hỗn hợp. Trong thế giới lý tưởng, $\mathcal{S}$ trả về **fail** cho $\mathcal{F}_{ns}$, trả về **fail** cho *C*.

Ret 2(b): (*id*, *pk*) *hiện đã được đăng ký nhưng bản ghi cuối cùng liên quan đến id là bản ghi thu hồi:* Các đối số cho trường hợp nhỏ này giống như Ret2(a), ngoại trừ một máy trạm trung thực *C*, kiểm tra xem bản ghi cuối cùng liên quan đến *id* có phải là bản ghi thu hồi hay không, sẽ trả về **fail**.

Ret 2(c): *id chưa được đăng ký nhưng bản ghi cuối cùng liên quan đến id là bản ghi đăng ký*: Trong thế giới hỗn hợp, một máy trạm *trung thực C* gửi (*sid*, **RetrieveDB**) tới $\mathcal{F}_{UDB}$. Nếu $\mathcal{A}$ trả về **allow** thì $\mathcal{F}_{UDB}$ sẽ gửi *DBstate* tới *C*, *C* thực hiện theo Bước 5a của giao thức π trong Hình 7, tính toán các bằng chứng W_1', W_2', W_3'. Sau đó, *C* gửi (*sid*, **Retrievestate**) tới $\mathcal{F}_{TP}$. Nếu $\mathcal{A}$ trả về **allow** thì *C* sẽ chạy Bước 5b của giao thức π. Do tính bảo mật của các bộ tích lũy c_1, c_2, ít nhất một trong các thuật toán xác minh sẽ trả về 0. Do đó, *C* trả về **fail**. Trong thế giới lý tưởng, $\mathcal{S}$, gửi **fail** tới $\mathcal{F}_{ns}$ và $\mathcal{F}_{ns}$ trả về **fail** cho *C*.

Ret 2(d): *id chưa được đăng ký nhưng bản ghi cuối cùng là bản ghi thu hồi*: Đối với trường hợp nhỏ này, chúng tôi có thể làm tương tự như Ret2(c), với điểm khác biệt là một máy trạm trung thực *C*, trả về **fail** khi nó kiểm tra xem bản ghi cuối cùng liên quan đến *id* là bản ghi thu hồi.

Chúng tôi bỏ qua các trường hợp môi trường $\mathcal{Z}$ gửi (*sid*, **VerifyID**, *id*) và (*sid*, **VerifyMapping**, *id*, *pk*) đến máy trạm *C* vì quá trình phân tích tương tự như trường hợp $\mathcal{Z}$ gửi (*sid*, **Retrieve**, *id*). Có thể dễ dàng nhận thấy điều này bằng cách mô tả Bước 6, 7 của giao thức π (Hình 7). Điều này hoàn thành chứng minh của chúng tôi. □

B Chứng minh Bổ đề 4.1

Bổ đề 4.1 ([30]). *Giả sử U là họ hàm Hash phổ quát 2 từ* $\{0,1\}^{3k}$ *đến* $\{0,1\}^{k}$. *Khi đó, với tất cả trừ một phân số* 2^{-k} *của các hàm* $f \in U$, *với mọi* $y \in \{0,1\}^{k}$, *một phân số của ít nhất* $1/ck$ *phần tử trong* $f^{-1}(y)$ *là các số nguyên tố trong đó c là một số nhỏ không thay đổi.*

Bổ đề 4.1 được chứng minh bằng dãy bổ đề. Chúng tôi suy ra từ chứng minh ở phần phụ lục của bài nghiên cứu [47].

Bổ đề B.1. *Cho* $U = \{ f : \{0,1\}^{m} \rightarrow \{0,1\}^{k} \}$ *là họ hàm Hash phổ quát 2.*

Với mọi $A \subseteq \{0,1\}^{m}$, *với tất cả trừ phân số* $O\left(\frac{2^{2k}}{|A|}\right)$ *của* $f \in U$, *nó đúng với mọi* $z \in \{0,1\}^{k}$, $\frac{|f^{-1}(z) \cap A|}{|f^{-1}(z)|} > \frac{|A|}{2^{m+1}}$.

Bổ đề B.1 được chứng minh bằng cách đầu tiên chỉ ra Bổ đề B.2 và sau đó là Bổ đề B.3.

Bổ đề B.2. *Cho $A \subseteq \{0,1\}^m$ và $z \in \{0,1\}^k$. Chúng ta nói $f \in U$ là (A, z)-cân bằng nếu* $\frac{2}{3} \cdot \frac{|A|}{2^k} \leq |f^{-1}(z) \cap A| \leq \frac{4}{3} \cdot \frac{|A|}{2^k}$. *Nó cho rằng* $\mathbf{Pr}_{f \in U}[f \text{ không phải } (A, z)\text{-cân bằng}] \leq \frac{9 \cdot 2^k}{|A|}$.

Chứng minh. Giả sử $A = \{a_1, \ldots, a_\ell\}$ và chúng tôi chọn $f \in U$ một cách ngẫu nhiên. Vì U là họ hàm Hash phổ quát 2 (Định nghĩa 3.2) nên chúng ta có $\Pr[f(a_i) = z] = \frac{1}{2^k}$.

Chúng tôi xác định biến ngẫu nhiên X_i bằng 1 nếu $f(a_i) = z$ và bằng 0 nếu ngược lại. Do đó, chúng ta có $\mathbf{Pr}[X_i = 1] = \frac{1}{2^k}$. Nếu $X = \sum_{i=1}^{\ell} X_i$ thì có thể dễ dàng nhận thấy rằng $X = |f^{-1}(z) \cap A|$. Chúng ta cũng có $\mu = E[X] = \ell \cdot \mathbf{Pr}[f(a_i) = z] = \ell \cdot 2^{-k}$. Bây giờ chúng tôi sẽ sử dụng bất đẳng thức Chebychev

$$\mathbf{Pr}[|X - \mu| \geq t] \leq \frac{Var(X)}{t^2}.$$

Nếu $t = \mu/3$, chúng ta có

$$\mathbf{Pr}_{f \in U}\left[|X - \mu| \geq \frac{\mu}{3}\right] \leq \frac{9Var(X)}{\mu^2}. \tag{2}$$

Vì $X_i, \ldots, X_\ell$ độc lập từng cặp nên ta có $Var(X) = \sum_{i=1}^{\ell} Var(X_i)$. Vì vậy, $Var(X) = \sum_{i=1}^{\ell}\left(E\left[X_i^2\right] - E[X_i]^2\right) = \ell \frac{1}{2^k}\left(1 - \frac{1}{2^k}\right) \leq \ell 2^{-k}$. Do đó, $\frac{9Var(X)}{\mu^2} \leq \frac{9 \cdot 2^k}{\ell}$. Do đó, theo (2),

$$\frac{2}{3} \cdot \frac{|A|}{2^k} \leq |f^{-1}(z) \cap A| < \frac{4}{3} \cdot \frac{|A|}{2^k}, \tag{3}$$

ngoại trừ phân số $\frac{9 \cdot 2^k}{\ell}$ của hàm $f \in U$. □

Bổ đề B.3. *Cho $A \subseteq \{0,1\}^m$, $z \in \{0,1\}^k$ và $f \in U$. Chúng ta nói rằng cặp (f, z) là "xấu" đối với tập hợp A nếu* $\frac{|f^{-1}(z) \cap A|}{|f^{-1}(z)|} \leq \frac{|A|}{2^{m+1}}$. *Khi đó, với mọi $A \subseteq \{0,1\}^m$, $z \in \{0,1\}^k$,*

$$\mathbf{Pr}_{f \in U}[(f, z) \text{ là "xấu" với } A] \leq \frac{18 \cdot 2^k}{|A|}. \tag{4}$$

Chứng minh. Chúng tôi cố định $z \in \{0,1\}^k$. Khi đó, theo Bổ đề B.2, nếu $A = \{0,1\}^m$ thì chúng ta có

$$\frac{2}{3} \cdot 2^{m-k} < |f^{-1}(z)| < \frac{4}{3} \cdot 2^{m-k}, \tag{5}$$

ngoại trừ phân số $9 \cdot 2^{k-m}$ của hàm $f \in U$. Tiếp theo, theo Bổ đề B.2 với $A \subseteq \{0,1\}^m$ tùy ý, ta có

$$\frac{2}{3}|A| \cdot 2^{-k} < |f^{-1}(z) \cap A| < \frac{4}{3}|A| \cdot 2^{-k}, \tag{6}$$

ngoại trừ phân số $\frac{9 \cdot 2^k}{|A|}$ của hàm $f \in U$.

Kết hợp (5), (6) ta có

$$\frac{|f^{-1}(z) \cap A|}{|f^{-1}(z)|} > \frac{|A|}{2^{m+1}}, \tag{7}$$

ngoại trừ phân số $\frac{18 \cdot 2^k}{|A|}$ của hàm $f \in U$. □

Theo Bổ đề B.3, sử dụng ràng buộc hỗn hợp, chúng ta có được Bổ đề B.1.

Chứng minh bổ đề 4.1. Bây giờ, cho *A* là tập hợp các số nguyên tố nhỏ hơn 2^m và $m = 3k$, vì Bổ đề 4.1 xem xét một họ hàm Hash phổ quát 2 từ $\{0, 1\}^{3k}$ đến $\{0, 1\}^k$. Theo định lý số nguyên tố, chúng ta có số nguyên tố nhỏ hơn hoặc bằng *x*, ký hiệu là $\pi(x)$, là tiệm cận $\frac{x}{\ln x}$. Bằng cách sử dụng một giới hạn không tiệm cận ([45]), chúng ta có điều đó cho bất kỳ $x \geq 55$, $\pi(x) > \frac{x}{\ln x+2}$. Vì vậy, chúng ta có

$$|A| = \pi\left(2^{3k}\right) > \frac{2^{3k}}{\ln 2^{3k}+2}. \tag{8}$$

Theo (8) và Bổ đề B.1, nó cho rằng ngoại trừ phân số $(\frac{18(3k+2\log_2 e)}{2^k \log_2 e})$ của các hàm $f \in U$,

$$\frac{\left|f^{-1}(z) \cap A\right|}{\left|f^{-1}(z)\right|} > \frac{1}{\frac{6k}{\log_2 e}+2} \approx \frac{1}{4 \cdot 16k+2}. \tag{9}$$

□

Người dịch: Nguyễn Văn Tú
Telegram: http://t.me/Tulibra
Link gốc: https://iohk.io/en/research/library/papers/towards-a-smart-contract-based-decentralized-public-key-infrastructure/