Cải thiện từ điển được xác thực động, với các ứng dụng cho Crypto*

Leonid Reyzin † Dmitry Meshkov ‡ Alexander Chepurnoy § Sasha Ivanov ¶

Ngày 2 tháng 4 năm 2017

Tóm lược

Chúng tôi cải thiện thiết kế và triển khai các từ điển được xác thực động của hai bên, ba bên và áp dụng các từ điển này cho sổ cái Crypto.

Sổ cái công khai (Blockchain) trong Crypto cần phải dễ dàng kiểm chứng. Tuy nhiên, việc duy trì cấu trúc dữ liệu của tất cả số dư tài khoản để xác minh xem giao dịch có hợp lệ hay không có thể khá nặng nề: trình xác minh (Verifier) không có lượng RAM lớn cần thiết cho cấu trúc dữ liệu sẽ hoạt động chậm do cần phải liên tục truy cập bộ nhớ thứ cấp. Chúng tôi chứng minh bằng thực nghiệm rằng từ điển được xác thực động có thể giảm đáng kể trọng tải cho trình xác minh. Mặt khác, bằng chứng cho mỗi giao dịch được tạo bởi từ điển xác thực làm tăng kích thước của Blockchain, điều này thúc đẩy chúng tôi tìm ra giải pháp với hầu hết các bằng chứng nhỏ gọn.

Những cải tiến của chúng tôi đối với thiết kế từ điển xác thực giúp giảm kích thước bằng chứng và tăng tốc độ xác minh lên 1,4–2,5 lần, giúp chúng phù hợp hơn cho ứng dụng Crypto. Chúng tôi tiếp tục chỉ ra rằng bằng chứng cho nhiều giao dịch trong một Block có thể được nén lại với nhau, giảm tổng độ dài của chúng xuống khoảng 2 lần.

Chúng tôi mô phỏng xác minh Blockchain và cho thấy rằng trình xác minh của chúng tôi có thể nhanh hơn khoảng 20 lần so với trình xác minh trên ổ đĩa dưới tải giao dịch thực tế.

1 Giới thiệu

Ứng dụng tạo động lực. Nhiều loại Crypto, bắt đầu bằng Bitcoin [Nak08], dựa trên sổ cái công khai của toàn bộ chuỗi tất cả các giao dịch đã từng diễn ra. Các giao dịch được xác minh và thêm vào sổ cái này bởi các Node được gọi là *thợ đào*. Nhiều giao dịch được nhóm thành các Block trước khi được thêm vào và sổ cái trở thành một chuỗi các Block như vậy, thường được gọi là *Blockchain*.

Nếu một thợ đào thêm một Block chứa giao dịch vào Blockchain, những thợ đào khác sẽ xác minh rằng mọi giao dịch đều hợp lệ và được ghi lại chính xác trước khi chấp nhận Block mới. (Những thợ đào cũng thực hiện các công việc khác để đảm bảo thỏa thuận chung trên Blockchain mà chúng tôi không đề cập ở đây.) Tuy nhiên, không phải chỉ những thợ đào mới tham gia vào một loại Crypto; những người khác xem

---

* Đây là phiên bản đầy đủ của bài nghiên cứu xuất hiện tại Hội nghị quốc tế lần thứ 21 về mật mã tài chính và bảo mật dữ liệu, tháng 4 năm 2017.

† Đại học Boston, http://www.cs.bu.edu/faculty/reyzin. Nghiên cứu được hỗ trợ bởi nền tảng Waves.

‡ Nghiên cứu IOHK, dmitry.meshkov@iohk.io

§ Nghiên cứu IOHK, alex.chepurnoy@iohk.io

¶ Nền tảng Waves, sasha@wavesplatform.com

Blockchain và/hoặc thực hiện xác minh một phần (ví dụ: đối với Node nhẹ, chẳng hạn như Node SPV của Bitcoin [Nak08, Phần 8]). Điều mong muốn là những người tham gia khác có thể kiểm tra một Blockchain với các đảm bảo bảo mật đầy đủ trên các thiết bị phần cứng thông thường, vì lợi ích của chính họ và vì việc duy trì một số lượng lớn các Node thực hiện xác thực đầy đủ là rất quan trọng đối với sức khỏe của Crypto [Par15]. Để xác minh từng giao dịch, chúng cần biết số dư tài khoản của người trả tiền.

Giải pháp đơn giản là yêu cầu mọi trình xác minh duy trì cấu trúc dữ liệu từ điển động gồm các cặp (khóa, giá trị), trong đó khóa là địa chỉ tài khoản (thường là khóa chung) và giá trị là số dư tài khoản. Thật không may, khi cấu trúc dữ liệu này phát triển, trình xác minh cần đầu tư nhiều vào RAM hơn (do đó không thể hoạt động với phần cứng thông thường nữa) hoặc chấp nhận sự chậm lại đáng kể đi kèm với việc lưu trữ cấu trúc dữ liệu trong bộ lưu trữ thứ cấp. Những sự chậm lại này (đặc biệt là những thứ gây ra bởi thời gian tìm kiếm ổ đĩa dài trong một tập hợp giao dịch được tạo ra một cách bất lợi) đã bị khai thác bởi các cuộc tấn công từ chối dịch vụ chống lại Bitcoin [Wik13] và Ethereum [But16].

Từ điển xác thực để giải quyết vấn đề. Chúng tôi đề xuất sử dụng các cấu trúc dữ liệu được xác thực bằng mật mã để giúp việc *xác minh* các giao dịch trong Blockchain rẻ hơn nhiều so với việc *thêm* chúng vào Blockchain. Xác minh rẻ hơn không chỉ mang lại lợi ích cho trình xác minh mà còn cho cả thợ đào: trong hệ thống Blockchain có nhiều loại Token (ví dụ: Token có thể đại diện cho các loại tiền tệ hoặc hàng hóa khác nhau), chẳng hạn như Nxt [nxt], thợ đào có thể chọn chỉ xử lý giao dịch cho một số loại Token, nhưng vẫn cần xác minh tất cả các giao dịch.

Cụ thể, chúng tôi đề xuất lưu trữ thông tin số dư trong *từ điển được xác thực động*. Trong cấu trúc dữ liệu như vậy, *trình chứng minh (Prover)* (trong trường hợp của chúng tôi là thợ đào) nắm giữ toàn bộ cấu trúc dữ liệu và sửa đổi nó khi các giao dịch được xử lý, xuất bản *bằng chứng* rằng mỗi giao dịch dẫn đến sửa đổi chính xác cấu trúc dữ liệu (những bằng chứng này sẽ được bao gồm với Block ghi lại giao dịch). Ngược lại, *trình xác minh* chỉ nắm giữ một *thông báo ngắn* về cấu trúc dữ liệu, xác minh bằng chứng và tính toán thông báo mới tương ứng với trạng thái mới của cấu trúc dữ liệu mà không cần phải lưu trữ chính cấu trúc đó. Chúng tôi nhấn mạnh rằng với cấu trúc dữ liệu được xác thực, trình xác minh có thể thực hiện các kiểm tra và cập nhật này mà không cần tin tưởng trình chứng minh: thuật toán xác minh sẽ từ chối mọi nỗ lực của trình chứng minh độc hại hoặc người trung gian cố ý đánh lừa trình xác minh chấp nhận kết quả không chính xác hoặc thực hiện sửa đổi không chính xác. Trái ngược với trường hợp không được xác thực đã thảo luận ở trên, trong đó trình xác minh phải lưu trữ toàn bộ cấu trúc dữ liệu, ở đây, bộ lưu trữ của trình xác minh là tối thiểu: 32 byte đủ cho một thông báo (ở mức bảo mật 128 bit), trong khi mỗi bằng chứng chỉ dài vài trăm byte và có thể bị loại bỏ ngay sau khi xác minh.

1.1 Đóng góp của chúng tôi

Cấu trúc dữ liệu từ điển được xác thực tốt hơn. Bởi vì giảm kích thước khối là mối quan tâm chính đối với các hệ thống Blockchain [CDE+16, DW13], chúng tôi tập trung vào việc giảm độ dài của bằng chứng sửa đổi, phải được đưa vào Block cho mỗi giao dịch. Ngoài ra, vì không có trọng tài trung tâm trong mạng Blockchain, chúng tôi yêu cầu cấu trúc dữ liệu được xác thực có thể hoạt động mà không có bất kỳ giả định nào về sự tồn tại của tác giả hoặc thiết lập đáng tin cậy và không có bất kỳ khóa bí mật nào (ví dụ: không giống như [PTT16, BGV11, CF13, CLH+15, MWMS16, CLW+16]). Bởi vì những thợ đào có thể có động lực để làm cho việc xác minh tốn nhiều thời gian hơn đối với những người khác, nên chúng tôi ưu tiên các cấu trúc dữ liệu có hiệu suất độc lập với các lựa chọn của thợ đào.

Chúng tôi thiết kế và triển khai cấu trúc dữ liệu từ điển được xác thực không yêu cầu thiết lập hoặc quyền tác giả đáng tin cậy có bằng chứng trung bình ngắn hơn 1,4 lần so với Skiplist (một loại cấu trúc dữ

liệu) được xác thực của [PT07] và ngắn hơn 2,5 lần so với cây đỏ đen (Red-Black Tree) của [CW11]. Hơn nữa, thời gian của trình chứng minh và trình xác minh của chúng tôi nhanh hơn bởi cùng một yếu tố so với thời gian tương ứng cho các Skiplist được xác thực và, không giống như công việc của [PT07], cấu trúc dữ liệu của chúng tôi là xác định, không cho phép trình chứng minh thiên vị các lựa chọn được cho là ngẫu nhiên để thực hiện hiệu suất tồi tệ hơn cho trình xác minh. Trên thực tế, hiệu suất *trong trường hợp xấu nhất* của cấu trúc dữ liệu của chúng tôi tương đương với hiệu suất *trong trường hợp dự kiến* của [PT07]. Công việc của chúng tôi được lấy cảm hứng từ các cây Merkle động [Mer89] của [NN00, AGT01, CW11, MHKS14] kết hợp với thuật toán cân bằng cây cổ điển của [AVL62].

Chúng tôi tiếp tục giảm thời lượng bằng chứng cho mỗi thao tác khi tập hợp các bằng chứng cho nhiều thao tác. Ví dụ: khi các bằng chứng cho 1000 thao tác trên từ điển 1.000.000 mục nhập được đặt cùng nhau, độ dài bằng chứng của chúng tôi giảm gần một nửa.

Cài đặt của chúng tôi về cấu trúc dữ liệu được xác thực, trong đó trình xác minh có thể tính toán thông báo mới sau khi sửa đổi thường được gọi là trường hợp "hai bên" (vì chỉ có hai bên: trình chứng minh và trình xác minh). Không nên nhầm lẫn với trường hợp "ba bên" dễ dàng hơn được đề cập trong nhiều tài liệu [Mer89, NN00, GT00, GTS01, AGT01, MND+04, GPT07, CW11], trong đó trình xác minh chỉ đơn giản được cung cấp thông báo mới sau khi sửa đổi (ví dụ: bởi chủ sở hữu dữ liệu đáng tin cậy). Mặc dù chúng tôi thiết kế chủ yếu cho trường hợp hai bên, kết quả của chúng tôi cũng có thể được sử dụng trong trường hợp ba bên, chẳng hạn như có thể thay thế Skiplist đã được xác thực của [GTS01] trong cả ứng dụng của hai bên và ba bên dựa trên chúng (ví dụ: [BP07, GPTT08, HPPT08, EK13] và nhiều loại khác), cải thiện hiệu suất và loại bỏ nhu cầu chọn ngẫu nhiên.

Ứng dụng cho Blockchain. Chúng tôi xem xét một hệ thống Blockchain đa Token (không giống như Bitcoin, chỉ có Bitcoin là Token) với các tài khoản trong đó số dư có thể tăng hoặc giảm theo thời gian (một lần nữa, không giống như Bitcoin, trong đó đầu ra giao dịch phải được chi tiêu cùng một lúc). Một ví dụ về hệ thống như vậy là Nxt [nxt]. Đối với mỗi Token t, có một cấu trúc dữ liệu được xác thực S_t duy trì số dư của tất cả tài khoản, được lưu trữ cục bộ bởi những thợ đào quan tâm đến khả năng thêm giao dịch cho loại Token đó. Tất cả thợ đào, bất kể sở thích, đều duy trì một bản sao cục bộ của thông báo ngắn về S_t.

Để tạo một Block có một số giao dịch, thợ đào sẽ thêm vào Block bằng chứng về tính hợp lệ của các giao dịch này, bao gồm bằng chứng về các cập nhật chính xác cho S_t và cũng bao gồm thông báo mới của S_t vào tiêu đề Block. Tất cả thợ đào, cũng như trình xác minh, xác minh bằng chứng liên quan đến thông báo mà họ biết và kiểm tra xem thông báo mới trong tiêu đề Block có chính xác không. (Điều quan trọng cần lưu ý là xác minh giao dịch bao gồm các bước khác không liên quan đến cấu trúc dữ liệu, chẳng hạn như xác minh chữ ký của người thanh toán trên giao dịch; các bước này không thay đổi.) Ngược lại với các Node xác minh thanh toán đơn giản [Nak08, Phần 8] bằng Bitcoin, những người không thể xác minh đầy đủ tính hợp lệ của một Block mới vì chúng không lưu trữ tất cả các đầu ra chưa chi tiêu, những trình xác minh có thể làm như vậy mà không lưu trữ bất kỳ thông tin số dư nào.

Mặc dù đã có nhiều đề xuất sử dụng cấu trúc dữ liệu được xác thực cho các Blockchain (xem ví dụ: [Tod16], [Mil12] và các tài liệu tham khảo trong đó), nhưng không nhiều đề xuất xuất bản bằng chứng cho các sửa đổi đối với cấu trúc dữ liệu. Ở cấp độ cao, cách tiếp cận của chúng tôi tương tự (nhưng hiệu quả hơn đáng kể so với) đề xuất của White [Whi15], người đề xuất xây dựng cấu trúc dữ liệu được xác thực dựa trên Trie (Trie là một cấu trúc cây dữ liệu được sử dụng để lưu trữ và truy vấn dữ liệu theo cách hiệu quả) cho Bitcoin (mặc dù anh ta không sử dụng các thuật ngữ đó).

Bởi vì cấu trúc dữ liệu được xác thực được cải tiến của chúng tôi, các trình chứng minh[1] và trình xác minh hiệu quả hơn và bằng chứng ngắn hơn so với các giải pháp trước đây. Chúng tôi cho thấy rằng bất cứ khi nào một Block bao gồm nhiều giao dịch cho một Token nhất định, bằng chứng của chúng có thể được kết hợp, giúp giảm thêm dung lượng sử dụng cho mỗi giao dịch, khoảng 2 lần đối với các điều kiện thực tế. Chúng tôi đánh giá quá trình xác minh tạo Block và chứng minh rằng việc xác minh cấu trúc dữ liệu được xác thực có thể nhanh hơn khoảng 20 lần so với việc duy trì cấu trúc dữ liệu chưa được xác thực đầy đủ trên ổ đĩa, trong khi việc tạo bằng chứng không làm tăng thêm nhiều vào tổng chi phí của thợ đào.

Giảm chi phí thiết lập ban đầu của thợ đào. Một thợ đào mới Molly muốn tham gia mạng phải tải xuống toàn bộ Blockchain và xác minh tính hợp lệ của mọi Block bắt đầu từ Block đầu tiên (được gọi là "khởi tạo"). Không cần thiết phải xác minh tính hợp lệ của mọi giao dịch, vì sự hiện diện của Block trong Blockchain đảm bảo với Molly rằng mỗi giao dịch đã được xác minh bởi các thợ đào khác khi Block được thêm vào. Tuy nhiên, nếu không có cấu trúc dữ liệu được xác thực, Molly vẫn cần tải xuống và phát lại tất cả các giao dịch để thiết lập số tiền cập nhật được giữ trong mỗi tài khoản và có thể xác thực các giao dịch trong tương lai.

Giải pháp của chúng tôi cho phép Molly giảm chi phí giao tiếp, tính toán và bộ nhớ khi tham gia mạng, bằng cách cho phép cô ấy tải xuống không phải toàn bộ Block với danh sách dài các giao dịch của chúng, mà chỉ các tiêu đề Block, ngoài việc chứng minh rằng Block đã được được tạo và liên kết chính xác với chuỗi, chứa thông báo tóm tắt của tất cả các giao dịch được xử lý và thông báo của mọi cấu trúc dữ liệu được xác thực S_t đã thay đổi kể từ Block trước đó. Thông tin này đủ để bắt đầu xác thực các giao dịch trong tương lai. Nếu Molly không chỉ muốn xác thực mà còn xử lý các giao dịch cho các Token loại *t*, thì cô ấy cần lấy toàn bộ S_t; tuy nhiên, điều quan trọng là cô ấy không cần một nguồn đáng tin cậy cho dữ liệu này, bởi vì cô ấy có thể xác minh tính chính xác của S_t so với thông báo[2].

2 Mô hình cho từ điển xác thực hai bên

Với sự đa dạng của các mô hình bảo mật cho các cấu trúc dữ liệu được xác thực, chúng tôi sẽ giải thích ngắn gọn về mô hình (theo hiểu biết tốt nhất của chúng tôi, nó được giới thiệu lần đầu tiên trong [BEG+91] và rõ ràng hơn trong [GSTW03, PT07]; nó thường được gọi là mô hình *hai bên*; xem [Pap11] để biết tổng quan về các tài liệu liên quan).

Mỗi trạng thái của cấu trúc dữ liệu được liên kết với một *thông báo* có thể tính toán hiệu quả; không thể tính toán được để tìm hai trạng thái khác nhau của cấu trúc dữ liệu tương ứng với cùng một thông báo. Có hai bên là *trình chứng minh* và *trình xác minh*. Trình chứng minh sở hữu cấu trúc dữ liệu, thực hiện các thao tác trên đó và gửi *bằng chứng* về các hoạt động này cho trình xác minh, người chỉ sở hữu bản tóm tắt trạng thái hiện tại của cấu trúc dữ liệu, có thể sử dụng bằng chứng để lấy kết quả của hoạt động và cập nhật

[1]Mức độ hiệu quả của việc tạo bằng chứng phụ thuộc vào thiết kế Crypto. Trong những loại Crypto mà mọi thợ đào đều cố gắng tạo một Block (chẳng hạn như Bitcoin), điều đó rất quan trọng, bởi vì mọi thợ đào đều phải chạy quy trình tạo bằng chứng. Mặt khác, trong những loại Crypto mà thợ đào giành được quyền tạo Block trước khi Block được tạo ra (chẳng hạn như Block dựa trên bằng chứng cổ phần [BGM16, KKR + 16]) , chỉ có một.

công cụ khai thác trên mỗi Block sẽ tạo ra bằng chứng.

[2]Ethereum [Woo14] bổ sung thông báo về trạng thái hiện tại của hệ thống cho mỗi Block, nhưng vì nó không triển khai bằng chứng cho việc sửa đổi cấu trúc dữ liệu nên thông báo này không thể được sử dụng trừ khi thợ đào tải xuống toàn bộ trạng thái của hệ thống, mặc dù quan trọng là trạng thái này có thể được tải xuống từ một nguồn không đáng tin cậy và được xác minh dựa trên thông báo. Miller và cộng sự [MHKS14, Phụ lục A] đã đề xuất sử dụng cấu trúc dữ liệu được xác thực để cải thiện việc sử dụng bộ nhớ, chứ không phải thời gian giao tiếp hoặc tính toán, của thiết lập ban đầu của Bitcoin.

thông báo của họ khi cấu trúc dữ liệu được sửa đổi. Mục tiêu bảo mật là đảm bảo các trình xác minh độc hại không bao giờ có thể đánh lừa các trình xác minh chấp nhận kết quả không chính xác hoặc tính toán các bản tóm tắt không chính xác. Điều quan trọng là không bên nào tạo ra hoặc sở hữu bất kỳ bí mật nào.

Mục tiêu bảo mật phụ (để ngăn chặn các cuộc tấn công từ chối dịch vụ của các trình chứng minh có thể có nhiều tài nguyên máy tính hơn trình xác minh) là để đảm bảo rằng một trình chứng minh độc hại không thể tạo ra các bằng chứng (dù hợp lệ hay không) khiến trình xác minh mất nhiều thời gian hơn để xử lý so với giới hạn trên được chỉ định trước.

Điều quan trọng là mô hình giả định rằng trình xác minh và trình chứng minh đồng ý về các hoạt động cấu trúc dữ liệu nào cần được thực hiện (trong ứng dụng Crypto của chúng tôi, các hoạt động sẽ đến từ các giao dịch và trình xác minh sẽ kiểm tra xem bản thân các hoạt động đó có hợp lệ hay không, ví dụ, kiểm tra chữ ký và số dư tài khoản của người trả tiền). Trình xác minh không được bảo vệ nếu nó thực hiện một thao tác khác với thao tác của trình chứng minh, bởi vì trình xác minh vẫn có thể tính toán một thông báo mới hợp lệ; nó sẽ chỉ nhận thấy sự khác biệt này nếu nó có thể thấy rằng bản tóm tắt mới của nó khác với bản tóm tắt mới của trình chứng minh. Mô hình cũng giả định rằng trình xác minh ban đầu có thông báo chính xác (ví dụ: bằng cách duy trì nó liên tục bắt đầu với trạng thái trống ban đầu của cấu trúc dữ liệu).

Cấu trúc dữ liệu cụ thể mà chúng tôi muốn triển khai là một từ điển (còn được gọi là bản đồ): nó cho phép chèn các cặp (khóa, giá trị) (đối với khóa mới), tra cứu giá trị theo khóa, cập nhật giá trị cho một giá trị khoá nhất định và xóa theo khoá.

Chúng tôi cung cấp các định nghĩa bảo mật chính thức trong Phụ lục A.

3 Các triển khai của chúng tôi

Mặc dù có rất nhiều công việc về cấu trúc dữ liệu được xác thực, nhưng theo hiểu biết tốt nhất của chúng tôi, chỉ có hai cấu trúc trước đó là các cấu trúc của [PT07] (dựa trên Skiplist) và [MHKS14] (dựa trên Skiplist và cây đỏ đen), giải quyết cài đặt chính xác của chúng tôi. Như đã đề cập trong phần giới thiệu, nhiều tài liệu khác đề cập đến cài đặt ba bên (mà chúng tôi cũng cải thiện), trong đó các sửa đổi được thực hiện bởi một tác giả đáng tin cậy và chỉ những tra cứu được thực hiện bởi những trình chứng minh tin tưởng vào thông báo. Một số tài liệu cũng đề xuất giải pháp yêu cầu khóa bí mật mà trình chứng minh vẫn chưa biết.

Chúng tôi sẽ giải thích các triển khai của chúng tôi từ quan điểm thống nhất công việc trước đó và áp dụng một số tối ưu hóa cho các ý tưởng hiện có.

Điểm xuất phát: Cây Merkle. Chúng tôi bắt đầu với cây Merkle cổ điển [Mer89]. Gọi H là hàm Hash chống va chạm. Các lá của cây lưu trữ dữ liệu mà chúng tôi muốn xác thực, trong trường hợp của chúng tôi là các cặp (khóa, giá trị). Nhãn của mỗi lá được định nghĩa là Hash của nội dung của nó (đứng trước một ký hiệu đặc biệt, ví dụ: 0 byte cho biết đó là một lá) và giá trị của mỗi Internal Node được xác định (đệ quy) là Hash của nhãn của hai phần tử con của nó (đứng trước một ký hiệu đặc biệt, ví dụ: 1 byte cho biết đó là một Internal Node). Thông báo là nhãn của gốc. Bằng chứng cho thấy một khóa đã cho nằm trong cấu trúc dữ liệu và có một giá trị nhất định bao gồm các nhãn của các Node cùng cấp trên đường dẫn từ gốc đến lá, cùng với thông tin về việc đường dẫn đi sang trái hay phải ở mỗi bước. Bằng chứng có thể được xác minh bằng cách tính toán lại nhãn được cho là nhãn gốc và kiểm tra xem nó có khớp với thông báo hay không. Bằng chứng này được gọi là *đường dẫn xác thực* .

Kết hợp cây tìm kiếm nhị phân. Để thực hiện tìm kiếm, chúng tôi biến cây thành một biến thể nhỏ của cây tìm kiếm nhị phân tiêu chuẩn, giống như trong [NN00, AGT01, MHKS14]. Đầu tiên, chúng tôi sắp xếp

các lá theo khóa. Mỗi Node nội (Internal Node) lưu trữ một khóa *y* là khóa nhỏ nhất của cây con bên phải của nó: theo cách đó, cây con bên trái chứa chính xác các lá có khóa nhỏ hơn *y*. (Mô tả này phá vỡ các ràng buộc theo cách ngược lại của [AGT01] và [MHKS14], nhưng trực quan hơn với những cải tiến của chúng tôi được mô tả bên dưới.) Không giống như cây tìm kiếm nhị phân tiêu chuẩn, cây tìm kiếm nhị phân này chỉ có các Internal Node để hỗ trợ tìm kiếm chứ không phải để lưu trữ các giá trị, chỉ ở các lá. Bằng chứng vẫn là cùng một đường dẫn xác thực. (Chúng tôi lưu ý rằng cách tiếp cận dựa trên cây tìm kiếm nhị phân tiêu chuẩn, trong đó các Internal Node cũng lưu trữ các khóa và giá trị, được khám phá trong [CW11]; như chúng tôi trình bày dưới đây trong Phần 4, nó dẫn đến các bằng chứng dài hơn, vì tiết kiệm được một chút chiều cao của cây bị phủ nhận nhiều hơn bởi thực tế là các Internal Node cũng phải bao gồm các khóa và giá trị của chúng vào tính toán nhãn của chúng và do đó đưa vào bằng chứng.)

Hơn nữa, chúng tôi đảm bảo rằng mọi Node không phải là lá đều có chính xác hai Node con. Để chèn một cặp (giá trị khóa) mới, hãy đi xuống đúng lá ℓ như trong cây tìm kiếm nhị phân tiêu chuẩn và thay thế ℓ bằng một Internal Node mới có ℓ và một lá mới chứa cặp (khóa, giá trị) mới như hai con của nó. Để đơn giản hóa việc chèn, chúng ta có thể đảm bảo rằng khóa được chèn luôn ở bên phải của ℓ và Internal Node mới nhận được cùng khóa với Node được chèn, bằng cách khởi tạo cây trống với một lá duy nhất chứa $-\infty$ như khóa (khi khóa là các giá trị ngẫu nhiên dài, chẳng hạn như khóa chung, việc đặt $-\infty$ thành chuỗi toàn 0 là hợp lý). (Thật dễ dàng để chứng minh rằng sau đó mọi phép chèn đều sang phải ở bước cuối cùng: nếu tìm kiếm cho một phép chèn không bao giờ đi một bước sang phải, thì nó đạt $-\infty$; và nếu nó đã đi một bước sang phải, sau đó xem xét khóa của Node cuối cùng khiến quá trình tìm kiếm thực hiện đúng bước và quan sát thấy khóa trong ℓ giống nhau và do đó nhỏ hơn khóa được chèn). Cách tiếp cận này giúp chúng ta tránh gặp phải các trường hợp đặc biệt trong mã Code và giảm một số phép so sánh cần thiết trong thao tác chèn.

Việc xóa phụ thuộc vào việc Internal Node *i* chứa khóa *k* bị xóa có hai con không phải là lá hay không: nếu vậy, chúng ta xóa lá ngoài cùng bên phải của cây con bên trái của *i* và sử dụng thông tin từ lá đó để ghi đè thông tin trong *i* và trong lá ngoài cùng bên trái trong cây con bên phải của *i*, là lá chứa *k*. Nếu *i* có một hoặc hai lá con, thì việc xóa rất dễ dàng, bởi vì bản thân *i* và lá con của nó có thể bị xóa; nếu con bên trái của *i* là một lá, thì thông tin của nó được sử dụng để ghi đè thông tin ở lá ngoài cùng bên trái của cây con bên phải của *i*.

Chứng minh sự vắng mặt của một khóa. Có hai cách tiếp cận để chứng minh tính không phải là thành viên của khóa *k* (đặc biệt cần thiết trong quá trình chèn). Cách tiếp cận đầu tiên (được sử dụng trong [NN00, AGT01, CW11, MHKS14]) là hiển thị bằng chứng cho hai lá lân cận có khóa $k_1 < k < k_2$. Cách tiếp cận thứ hai (được sử dụng trong [GT00] và các tài liệu khác dựa trên Skiplist, chẳng hạn như [PT07]) là thêm một con trỏ tiếp theo vào mỗi lá và sửa đổi cách tính nhãn của lá, bằng cách Hash không chỉ khóa và giá trị được lưu trữ tại lá, nhưng cũng là khóa của lá tiếp theo (và $+\infty$ khi không có lá tiếp theo).

Chúng tôi áp dụng cách tiếp cận thứ hai vì tính đơn giản của nó: nó thống nhất mã Code để tra cứu thành công và không thành công, trong cả hai trường hợp đều cung cấp cho chúng tôi bằng chứng bao gồm một đường dẫn xác thực duy nhất. (Mặc dù cách tiếp cận thứ hai này kéo dài bằng chứng của mỗi lần tra cứu thành công bằng độ dài của khóa, nhưng nó lại rút ngắn một chút bằng chứng về một lần tra cứu không thành công trung bình bằng khoảng độ dài của nhãn.). Ngoài ra, việc chúng tôi tạo ra một $-\infty$ giám sát, đảm bảo rằng các lần chèn luôn đi vào bên phải của một lá hiện có, làm cho việc duy trì con trỏ tới lá tiếp theo trở nên đơn giản: khi một lá ℓ_{new} được chèn vào bên phải của lá ℓ_{old}, chỉ cần đặt ℓ_{new}.**next** = ℓ_{old}.**next** và ℓ_{old}.**next** = ℓ_{new}. Việc xóa cũng cần đảm bảo rằng con trỏ này được duy trì, bằng cách đi tới phần trước của lá bị xóa (do đó, việc xóa luôn chạm vào hai lá, bất kể vị trí của khóa bị xóa trong cây).

Cập nhật giá trị cho khóa hiện tại. Nếu trình chứng minh cập nhật giá trị được lưu trữ tại một lá (ví dụ: trừ đi số tiền được sử dụng cho một giao dịch), nhãn của lá đó và tất cả các Node phía trên nó cần được tính

toán lại, nhưng không có thông tin nào khác trong cây thay đổi. Quan sát rằng việc tính toán lại các nhãn này chỉ cần giá trị mới ở lá và thông tin đã có trong đường dẫn xác thực. Do đó, trình xác minh có tất cả thông tin cần thiết để tính toán thông báo mới sau khi kiểm tra xem đường dẫn xác thực có đúng không. Do đó, bằng chứng cho một bản cập nhật cũng giống như bằng chứng cho một tra cứu.

Chèn đơn giản. Các phần chèn vào cây tìm kiếm nhị phân Merkle của chúng tôi, giống như các phần chèn vào cây tìm kiếm nhị phân thông thường, có thể yêu cầu một số cân bằng lại để đảm bảo rằng các đường dẫn đến các lá không phát triển quá dài, làm tăng thời gian tính toán và giao tiếp trên mỗi thao tác. Tuy nhiên, chúng ta sẽ thảo luận về tái cân bằng trong phần tiếp theo. Hiện tại, hãy xem xét việc chèn mà không cần cân bằng lại. Việc chèn như vậy chỉ đơn giản là thay thế một lá cũ (được tìm thấy bằng cách thực hiện tìm kiếm khóa được chèn) bằng một Internal Node mới, được liên kết với hai lá. Do đó, kiến thức về nội dung của hai lá này và đường dẫn xác thực là đủ để có thể tính toán thông báo mới. Do đó, bằng chứng cho việc chèn đơn giản như vậy vẫn giống như trước đây: đường dẫn xác thực đến lá được tìm thấy trong quá trình tìm kiếm khóa đã được chèn. Bằng chứng này đủ để trình xác minh kiểm tra xem khóa không tồn tại và thực hiện việc chèn.

3.1 Cải tiến của chúng tôi

Quan sát 1: Sử dụng các hoạt động cân bằng cây luôn đi trên đường dẫn. Có nhiều thuật toán để cân bằng cây tìm kiếm nhị phân. Ở đây chúng tôi tập trung vào các cây AVL [AVL62], cây đỏ đen [GS78] (và biến thể có sự phân bổ Node đỏ ưu tiên về phía trái của chúng [Sed08]) và các Treap [SA96] (và các cây tìm kiếm nhị phân cân bằng ngẫu nhiên tương đương của chúng [MR98]). Tất cả chúng đều duy trì một số thông tin bổ sung trong các Node cho phép các thuật toán chèn và xóa đưa ra quyết định về việc có thực hiện xoay cây hay không và bằng cách nào để duy trì một cây cân bằng hợp lý. Chúng có thể dễ dàng được điều chỉnh để hoạt động với các cây được sửa đổi một chút của chúng tôi chỉ có các giá trị ở các lá (chỉ đơn giản là không áp dụng bất kỳ quy trình cân bằng nào cho các lá) và tất cả đều duy trì tính bất biến của chúng tôi rằng khóa của Internal Node là tối thiểu của cây con bên phải của nó ngay cả sau khi xoay.

Thông tin bổ sung mà chúng duy trì để cân bằng thường không lớn (chỉ một bit cho "màu" trên mỗi Node đối với cây đỏ đen; một trit cho "cân bằng" trên mỗi Node đối với cây AVL; và khoảng log n bit để lưu trữ "độ ưu tiên" trên mỗi Node đối với các Treap, trong đó n là số Node). Thông tin này sẽ được thêm làm đầu vào cho tính toán Hash cho nhãn của mỗi Internal Node. Thông tin này đối với mỗi Node trên đường dẫn từ gốc đến lá cũng nên được đưa vào bằng chứng (vì nó phải được trình xác minh nhập vào Hash).

Đối với các lần chèn, chúng tôi quan sát thấy rằng nếu hoạt động cân bằng cây chỉ xoay tổ tiên của lá mới được chèn và không sử dụng hoặc sửa đổi thông tin trong bất kỳ Node nào khác, thì bằng chứng mà chúng tôi đã cung cấp cho tìm kiếm có đủ thông tin để trình xác minh thực hiện thao tác chèn và cân bằng cây. Đây là trường hợp của cây AVL[3] và các Treap. Cây đỏ đen, tùy thuộc vào biến thể, có thể truy cập thông tin ở con và cháu của tổ tiên để quyết định phép xoay, do đó ít phù hợp hơn cho ứng dụng của chúng tôi, vì nội dung của những con và cháu đó sẽ cần phải được chứng minh, kéo dài bằng chứng. (Có thể sửa

[3]Đối với những người quen thuộc với cây AVL, chúng tôi lưu ý rằng đây là trường hợp khi cây AVL được triển khai với mọi Node duy trì sự khác biệt về độ cao giữa con phải và con trái, thay vì độ cao của chính nó, bởi vì nếu một Node duy trì độ cao thì nó cần phải tính toán sự cân bằng của nó để quyết định xem có xoay hay không và phép tính này yêu cầu chiều cao của cả hai con, trong khi bằng chứng của chúng tôi chỉ chứa chiều cao của một.

đổi cây đỏ đen bằng cách lưu trữ màu của các Node với cha mẹ và/hoặc ông bà, nhưng chúng tôi không khám phá tùy chọn này vì chúng tôi tìm thấy giải pháp tốt hơn.)

Tuy nhiên, trong số các tùy chọn này, chỉ có các cây đỏ đen được triển khai trong cài đặt của chúng tôi [MHKS14] và việc triển khai này đôi khi phải truy cập vào màu của một Node không phải là tổ tiên của lá mới được chèn. Do đó, bằng chứng chèn của [MHKS14] phải dài hơn, để bao gồm các đường dẫn xác thực đến các Node bổ sung (theo Miller [Mil16], bằng chứng cho việc chèn vào cây đỏ đen của [MHKS14] dài hơn khoảng 3 lần so với bằng chứng để tra cứu). Do đó, các cây cân bằng phù hợp hơn với bối cảnh của chúng tôi chưa từng được triển khai trước đây (hãy lưu ý rằng các Treap đã được triển khai trong bối cảnh ba bên của [CW11]; xem phần so sánh của chúng tôi trong Phần 4).

Để xóa, các hoạt động cân bằng cây đôi khi phải xem xét các cây cân bằng không tuân thủ đường dẫn chính mà chúng ta biết. May mắn thay, các cây AVL hoạt động rất tốt ngay cả khi xóa: trung bình, số lượng Node không tuân thủ đường dẫn chính cho mỗi lần xóa là ít hơn một, bởi vì tối đa hai Node không tuân thủ là cần thiết cho mỗi vòng xoay và có khoảng 0,26 vòng xoay mỗi lần xóa, theo thử nghiệm của chúng tôi (Knuth [Knu98, trang 474] đưa ra một con số thậm chí còn nhỏ hơn là 0,21).

Quan sát 2: Không Hash khóa nội (Internal Key). Để xác minh rằng có một lá cụ thể (đó là tất cả những gì chúng ta cần cho cả câu trả lời khẳng định và phủ định), trình xác minh không cần biết lá đó được tìm thấy như thế nào mà chỉ cần biết lá đó được kết nối với gốc thông qua một chuỗi Hash thích hợp. Do đó, giống như các tác giả của [PT07] (và nhiều tài liệu trong cài đặt ba bên), chúng tôi không thêm khóa của các Internal Node vào đầu vào Hash và không đưa chúng vào bằng chứng. Điều này trái ngược với công việc của [MHKS14], cách tiếp cận chung yêu cầu nhãn phụ thuộc vào toàn bộ nội dung của một Node và do đó yêu cầu gửi khóa của các Internal Node tới trình xác minh để trình xác minh có thể tính toán các nhãn. Khi các khóa không chiếm nhiều dung lượng (như trong [MHKS14]), sự khác biệt giữa việc gửi khóa của một Internal Node và gửi hướng (trái hoặc phải) mà đường dẫn tìm kiếm đã thực hiện là nhỏ. Tuy nhiên, khi các khóa có độ dài tương đương với nhãn (như trong ứng dụng Crypto, vì chúng là số nhận dạng tài khoản, được tính là đầu ra của Hash hoặc khóa công khai), sự khác biệt này có thể có nghĩa là gần bằng hệ số hai trong độ dài bằng chứng.

Quan sát 3: Skiplist chỉ là một biến thể của các Treap. Dean và Jones [DJ07] quan sát thấy rằng Skiplist [Pug90] tương đương với cây tìm kiếm nhị phân. Cụ thể, việc chuyển đổi từ Skiplist sang cây tìm kiếm nhị phân chỉ đơn giản là xây dựng một cây trên đỉnh tháp của Skiplist, tuân theo quy tắc rằng cấp độ của con thấp hơn (hoặc bằng, nếu con đúng) so với cha mẹ. Tất cả các Node không phải là đỉnh (nghĩa là các giá trị lặp lại) sau đó có thể bị xóa. Các giá trị gặp phải trong tìm kiếm sẽ giống nhau đối với Skiplist và cây kết quả. Chúng cho thấy rằng việc thêm vào Skiplist có thể được coi là tương đương với việc chèn vào cây: thay vì xây dựng một tòa tháp có cấp độ nhất định trong Skiplist, hãy chèn giá trị vào cây ở dưới cùng và xoay theo cấp độ của cấp độ con và cấp độ của cha mẹ thỏa mãn quy tắc trên. Chúng tôi giới thiệu người đọc đến [DJ07] để biết chi tiết.

Chúng tôi mở rộng quan sát của [DJ07] để lưu ý rằng các Skiplist được xem theo cách này chỉ là một biến thể của các lần xử lý, với "cấp độ" trong Skiplist dựa trên cây tương ứng với "độ ưu tiên" trong một lần xử lý. Các độ cao trong Skiplist được lấy mẫu sao cho giá trị h có xác suất $1/2^{h+1}$, trong khi các ưu tiên trong một Treap được lấy mẫu thống nhất, nhưng nếu không thì chúng tương đương nhau. Tất nhiên, chúng tôi tiếp tục chuyển đổi chế độ xem Skiplist dựa trên Treap này để chỉ có các giá trị tại các lá, như đã được mô tả ở trên. Chế độ xem này cho phép chúng tôi kiểm tra hiệu suất của Skiplist và Treap với cách triển khai cơ bản giống nhau.

Trong công việc trước đây, để làm cho chúng được xác thực, các Skiplist về cơ bản đã được chuyển đổi thành cây nhị phân bởi [GT00]; chuyển đổi này đã được thực hiện rõ ràng trong [CW11]. Cây nhị phân của

chúng tôi, kết quả là kết hợp việc quan sát [DJ07] với phép biến đổi chỉ đặt các giá trị ở các lá, cuối cùng gần như giống hệt nhau, với điểm khác biệt chính sau: cấu trúc dữ liệu của chúng tôi lưu trữ giá trị tối thiểu của cây con bên phải của mỗi Internal Node, trong khi cấu trúc dữ liệu của [GT00] lưu trữ giá trị nhỏ nhất của toàn bộ cây con của mỗi Internal Node. (Để xem sự tương đương, hãy lưu ý rằng cấu trúc dữ liệu của chúng tôi có thể được lấy từ cấu trúc dữ liệu của [PT07] bằng cách yêu cầu mọi cha mẹ thay thế khóa của nó bằng khóa của con bên phải của nó; điểm khác biệt duy nhất còn lại là chuỗi các lá không theo tiêu chuẩn trong Skiplist.) Tuy nhiên, không triển khai trước, Skiplist được xử lý giống như các cây nhị phân khác.

Quan sát 4: Tính xác định là tốt hơn. Các Treap và Skiplist hoạt động tốt như mong đợi khi các ưu tiên (đối với các Treap) và cấp độ (đối với Skiplist) được chọn ngẫu nhiên, độc lập với các khóa trong cấu trúc dữ liệu. Tuy nhiên, nếu một kẻ tấn công có thể tác động hoặc dự đoán các lựa chọn ngẫu nhiên, thì các đảm bảo về hiệu suất sẽ không còn nữa. Trong bối cảnh của chúng tôi, vấn đề là trình chứng minh và trình xác minh cần phải đồng ý bằng cách nào đó về tính ngẫu nhiên được sử dụng. (Đây không phải là vấn đề đối với cài đặt ba bên, trong đó tính ngẫu nhiên có thể được cung cấp bởi tác giả đáng tin cậy.)

Công việc trước đây trong mô hình ba bên đã đề xuất chọn mức độ ưu tiên và cấp độ bằng cách áp dụng hàm Hash cho các khóa [CW11, Phần 3.1.1]. Tuy nhiên, do các khóa được chèn có thể bị ảnh hưởng bởi kẻ tấn công, nên phương pháp tạo ngẫu nhiên này có thể mang lại cho kẻ tấn công khả năng làm cho cấu trúc dữ liệu rất chậm và thời gian kiểm chứng rất lâu, cho phép tấn công từ chối dịch vụ một cách hiệu quả. Để loại bỏ cuộc tấn công này bởi một kẻ thù bên ngoài, chúng ta có thể thêm chuỗi ngẫu nhiên vào hàm Hash sau khi các giao dịch được chọn để kết hợp vào cấu trúc dữ liệu (ví dụ: bao gồm một chuỗi ngẫu nhiên mới vào mỗi tiêu đề Block). Tuy nhiên, kẻ tấn công bên trong vẫn đưa ra vấn đề: trình chứng minh chọn loại chuỗi này và các giao dịch sẽ có khả năng làm cho cấu trúc dữ liệu trở nên kém hiệu quả hơn đối với mọi người bằng cách chọn một loại chuỗi xấu, vi phạm mục tiêu bảo mật phụ đã nêu trong Phần 2.

Quan sát 5: Cây AVL hoạt động tốt hơn ở các thông số phù hợp nhất. Bất kể phương pháp cân bằng cây nào (miễn là nó thỏa mãn các quan sát 1 và 2), chi phí tra cứu, cập nhật và chèn được xác định đơn giản bởi độ sâu của lá liên quan, bởi vì số lượng Node đi qua, kích thước của bằng chứng, và số lượng Hash được thực hiện bởi cả trình chứng minh và trình xác minh tỷ lệ thuận với độ sâu này. Tất nhiên, các phương pháp cân bằng cây khác nhau có thể sử dụng logic hơi khác nhau và gây ra số lần xoay khác nhau, nhưng lượng thời gian dành cho các phương pháp đó là không đáng kể so với chi phí đánh giá hàm Hash (lưu ý rằng một lần xoay cây chỉ thay đổi hai con trỏ và không thay đổi số lượng giá trị Hash cần được tính nếu cả cha và con đều trên đường dẫn đến lá).

Khoảng cách trường hợp trung bình giữa gốc và lá ngẫu nhiên cho cả cây AVL và cây đỏ đen sau khi chèn n khóa ngẫu nhiên rất gần với $\log_2 n$ tối ưu [Knu98, p. 468], [Sed08]. Khoảng cách trường hợp xấu nhất đối với cây đỏ đen gấp đôi khoảng cách tối ưu [Sed08], trong khi khoảng cách trường hợp xấu nhất đối với cây AVL gấp 1,44 lần khoảng cách tối ưu [Knu98, tr. 460]. Ngược lại, khoảng cách dự kiến (không phải trường hợp xấu nhất) cho các Treap và Skiplist là 1,5 lần khoảng cách tối ưu [Pug90]. Do đó, cây AVL, ngay cả trong *trường hợp xấu nhất*, tốt hơn so với các Treap và Skiplist *trong kỳ vọng*.

Quan sát 6: Bằng chứng cho nhiều hoạt động có thể được nén. Khi nhiều hoạt động trên cấu trúc dữ liệu được xử lý cùng nhau, bằng chứng của chúng có thể được nén. Trình xác minh sẽ không cần nhãn của bất kỳ Node nào nhiều hơn một lần. Hơn nữa, trình xác minh sẽ không cần nhãn của bất kỳ Node nào nằm trên đường dẫn đến lá trong một bằng chứng khác (vì nó sẽ được tính toán trong quá trình xác minh bằng chứng đó). Trình xác minh cũng sẽ không cần nhãn của bất kỳ Node nào được tạo bởi trình xác minh (ví dụ: nếu có sự chèn vào cây con bên phải của gốc, thì trình xác minh sẽ thay thế Node con bên phải của gốc bằng

một Node mới, do đó sẽ biết nhãn của nó khi nhãn đó là cần thiết cho bằng chứng về một số hoạt động tiếp theo trên cây con bên trái).

Việc thực hiện quá trình nén này là không cần thiết (thuật toán nén chung, như được sử dụng trong [MHKS14] và được [Mil16] báo cáo cho chúng tôi, có thể xử lý các nhãn lặp lại, nhưng sẽ không thực hiện các tối ưu hóa khác). Chúng tôi đề xuất cách tiếp cận sau để nén một loạt các hoạt động, về mặt khái niệm sẽ tách các Node của cây khỏi các hoạt động liên quan đến chúng.

Gọi *S* là cây ban đầu. Mọi Node của *S* được truy cập khi thực hiện loạt thao tác được đánh dấu là "đã truy cập". Các Node mới và các Node được sửa đổi không thay thế các Node của *S* mà được tạo mới và được đánh dấu là "mới". Do đó, *S* được giữ nguyên và các Node mới có thể trỏ đến các Node của *S* (nhưng các Node của *S* sẽ không trỏ đến các Node mới). Ở cuối lô, có một cây con của *S* (bắt đầu từ gốc) được đánh dấu là "đã truy cập" và một cây con của cây mới (bắt đầu từ gốc của cây mới) được đánh dấu là "mới".

Bằng chứng chứa nội dung của các Node của cây con "đã truy cập" này của *S* (không bao gồm các khóa cho các Internal Node nhưng bao gồm cả khóa và các khóa của lá tiếp theo cho các lá), cũng như nhãn của các Node cách cây con này một bước. Bằng chứng như vậy rất dễ lấy và sắp xếp theo thứ tự bằng cách thực hiện truyền tải theo thứ tự các Node "đã truy cập" và các Node con của chúng, đồng thời viết ra thông tin thích hợp về từng Node đạt được trong quá trình truyền tải. Ngoài cây này, bằng chứng còn chứa một chuỗi các "hướng" bit đơn (trái hoặc phải) (xem Quan sát 2) mà thuật toán của trình chứng minh đã thực hiện khi thực hiện loạt thao tác. Sau khi trình chứng minh xây dựng bằng chứng, các cờ "đã truy cập" và "mới" được đặt lại cho đợt tiếp theo (thông qua các lần duyệt của hai cây con) và các Node của *S* không thể truy cập được từ gốc cây mới có thể được xoá bỏ để tiết kiệm tài nguyên.

Trình xác minh chỉ cần xây dựng lại phần đã truy cập này của *S* bằng cách sử dụng bằng chứng và tính toán nhãn gốc của cây được xây dựng lại để đảm bảo rằng nó bằng với thông báo. Sau đó, trình xác minh về cơ bản chạy cùng một thuật toán như trình chứng minh để thực hiện hàng loạt sửa đổi. Sự khác biệt duy nhất là trình xác minh thay thế các phép so sánh khóa trên các Internal Node bằng cách đi theo hướng trái hoặc phải từ bằng chứng và thêm kiểm tra xem khóa được tìm có bằng với khóa trong lá hoặc giữa khóa trong lá và khóa của lá tiếp theo (việc kiểm tra này đảm bảo các hướng trong bằng chứng là trung thực).

Kết hợp những quan sát này lại với nhau, chúng tôi thu được cấu trúc dữ liệu để triển khai: một cây AVL với các giá trị chỉ được lưu trữ ở các lá, đôi khi được gọi là cây AVL+. Chúng tôi triển khai cấu trúc dữ liệu này và so sánh nó với các tùy chọn khác trong phần tiếp theo. Chúng tôi chứng minh tính bảo mật của nó trong Phụ lục B.

4 Thực hiện và Đánh giá

Chúng tôi đã triển khai cây AVL+, cũng như các Treap và Skiplist dựa trên cây của chúng tôi, bằng ngôn ngữ lập trình Scala [sca] bằng cách sử dụng hàm Hash Blake2b [ANWOW13] với đầu ra 256 bit (32 byte). Việc triển khai của chúng tôi có sẵn tại [cod][4]. Để triển khai AVL+, chúng tôi đã sử dụng mô tả trong sách giáo khoa [Wei06] với quy trình tính toán số dư giống như trong [Pfa02, Chương 5]. Chúng tôi đã chạy thử nghiệm bằng cách đo chi phí của 1000 lần chèn ngẫu nhiên (với khóa 26 byte và giá trị 8 byte) vào cấu trúc dữ liệu đã có kích thước $n = 0, 1000, 2000, \ldots, 999000$ khóa trong đó.

[4]Lưu ý rằng việc triển khai cây AVL+ với tính năng nén bằng chứng cho nhiều thao tác có đầy đủ tính năng, trong khi các triển khai khác (có trong thư mục con "cũ") là đủ để thực hiện các phép đo được báo cáo trong bài nghiên cứu này, nhưng thiếu các tính năng, chẳng hạn như xóa, xử lý lỗi và nén nhiều bằng chứng.

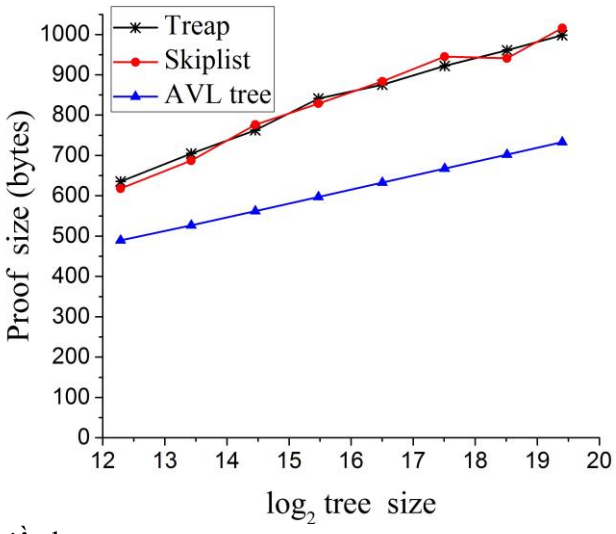
Đúng như dự đoán, độ dài của đường dẫn từ gốc đến một lá ngẫu nhiên trong cây AVL+ n lá chỉ kém hơn 2 - 3% so với $\log_2 n$ tối ưu. Ngược lại, độ dài của đường dẫn trong Skiplist thường kém hơn khoảng 44% so với tối ưu và trong Treap kém hơn khoảng 32% so với tối ưu.

Độ dài bằng chứng cho một hoạt động. Độ dài trung bình của bằng chứng chèn khóa mới vào cây 1.000.000 Node với Hash 32 byte, khóa 26 byte và giá trị 8 byte là 753 byte. Bây giờ chúng tôi giải thích con số này và so sánh nó với công việc trước đây.

Lưu ý rằng đối với đường dẫn có độ dài k, bằng chứng bao gồm:

- k nhãn (là giá trị Hash),
- k + 1 ký hiệu cho biết bước tiếp theo là phải hay trái hoặc chúng ta đang ở một chiếc lá không có bước tiếp theo (những ký hiệu này khớp với 2 bit mỗi ký tự),
- k mẩu thông tin về cân bằng hoặc mức (những mẩu này vừa với 2 bit cho cây AVL+, nhưng yêu cầu 1 byte cho Skiplist và 3 hoặc 4 byte cho các Treap),
- khóa lá, khóa lá tiếp theo và giá trị được lưu trữ trong Node lá (khóa lá không cần thiết trong bằng chứng cho việc tra cứu và cập nhật khóa hiện có, mặc dù kỹ thuật nén Quan sát 6 của chúng tôi sẽ bao gồm nó, bởi vì nó không theo dõi lý do tại sao đạt được một chiếc lá)

Do đó, độ dài bằng chứng gần như tỷ lệ thuận với độ dài đường dẫn: với Hash 32 byte, khóa 26 byte và giá trị 8 byte, bằng chứng chiếm 34k+61 byte giả sử chúng tôi không tối ưu hóa ở cấp độ bit hoặc khoảng k ít byte hơn nếu chúng tôi thực hiện (việc triển khai của chúng tôi hiện không có). Lưu ý rằng giá trị của k cho n = 1.000.000 là khoảng 20 đối với cây AVL+ và khoảng 29 đối với Skiplist, điều đó có nghĩa là bằng chứng dựa trên cây AVL ngắn hơn khoảng 1,4 lần so với bằng chứng dựa trên Skiplist. Bằng chứng Treap có k nhỏ hơn một chút, nhưng lợi thế này hoàn toàn bị phủ nhận trong các thử nghiệm của chúng tôi bởi các byte bổ sung cần thiết để ghi lại cấp độ.



Độ dài bằng chứng cho việc xóa có thể thay đổi nhiều hơn (vì thao tác xóa đi đến hai lá lân cận và cũng có thể cần các Node ngoài đường dẫn để xoay), nhưng trung bình lớn hơn 50 byte so với thao tác chèn, tra cứu và cập nhật.

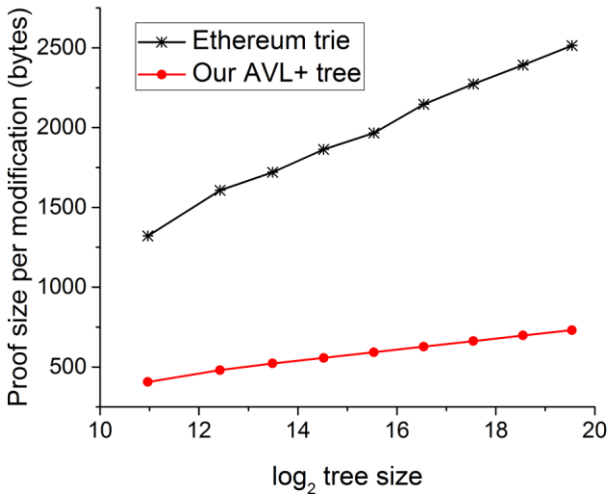
So sánh độ dài bằng chứng với công việc hiện có. Các con số của chúng tôi phù hợp với những con số được báo cáo bởi Papamanthou và Tamassia [PT07, Phần 4], những người cũng báo cáo các đường dẫn có độ dài 30 cho các Skiplist có 1.000.000 mục nhập. (Họ sử dụng hàm Hash kém an toàn hơn có độ dài đầu ra bằng một nửa của chúng tôi, dẫn đến bằng chứng ngắn hơn; nếu họ chuyển sang hàm Hash an toàn hơn, bằng chứng của họ sẽ có cùng độ dài với bằng chứng dựa trên Skiplist của chúng tôi, do đó lâu hơn 1,4 lần so với bằng chứng dựa trên AVL+ của chúng tôi).

So sánh trực tiếp với công việc của [MHKS14] thì khó hơn, bởi vì thông tin về độ dài bằng chứng cho một lần chèn vào cây đỏ đen không được cung cấp trong [MHKS14] (những gì được báo cáo trong [MHKS14] là kết quả của việc nén dữ liệu bằng phương pháp Gzip [GA] của phép nối các bằng chứng cho

100.000 thao tác tra cứu). Tuy nhiên, vì các khóa của các Internal Node được bao gồm trong bằng chứng của [MHKS14], nên bằng chứng cho tra cứu phải dài hơn khoảng 1,7 so với trong cây AVL+ của chúng tôi (đối với độ dài Hash và khóa của chúng tôi). Theo [Mil16], bằng chứng cho việc thêm vào cây đỏ đen của [MHKS14] dài hơn khoảng 3 lần so với tra cứu (và do đó dài hơn khoảng 5 lần so với bằng chứng cho việc chèn vào cây AVL+ của chúng tôi). Tất nhiên, công việc [MHKS14] có lợi thế là tổng quan, cho phép triển khai bất kỳ cấu trúc dữ liệu nào, bao gồm cả cây AVL+, điều này sẽ làm giảm chi phí chèn thêm vào chi phí tra cứu; nhưng tổng quan không tránh khỏi việc gửi các khóa nội, vì vậy chi phí tra cứu sẽ vẫn còn.

Chúng tôi cũng có thể so sánh công việc của mình với công việc trên cấu trúc dữ liệu được xác thực ba bên, vì cấu trúc dữ liệu của chúng tôi cũng hoạt động trong mô hình ba bên (nhưng không phải ngược lại: cấu trúc dữ liệu được xác thực ba bên không hoạt động trong mô hình của chúng tôi, bởi vì chúng hoạt động không cho phép trình xác minh tính toán thông báo mới, mặc dù một số có thể được điều chỉnh để làm như vậy). Công việc dựa trên Skiplist, chẳng hạn như [AGT01, GTS01, GPT07], có kích thước bằng chứng giống như [PT07] đã đề cập và do đó cải tiến của chúng tôi có cùng hệ số 1,4.

Đối với công việc ba bên dựa trên cây đỏ đen, có hai biến thể. Biến thể chỉ lưu trữ các giá trị ở lá, giống như chúng tôi, đã được triển khai bởi Anagnostopoulos và cộng sự [AGT01], người không báo cáo độ dài bằng chứng; tuy nhiên, chúng tôi có thể suy ra nó gần đúng từ số lần Hash được báo cáo trong [AGT01, Hình 6, "số lần Hash trên mỗi lần chèn"] và kết luận rằng nó kém hơn khoảng 10-20% so với của chúng tôi. Biến thể sử dụng cây tìm kiếm nhị phân tiêu chuẩn, với các khóa và giá trị trong mọi Node, được triển khai bởi [CW11] và có bằng chứng ngắn nhất trong số các cấu trúc dữ liệu được thử nghiệm trong [CW11]. Độ dài bằng chứng trung bình (đối với câu trả lời khẳng định) trong [CW11] là khoảng 1500 byte khi tìm kiếm khóa ngẫu nhiên trong cây bắt đầu trống và tăng lên 10^5 Node, với khóa, giá trị và Hash 28 byte. Ngược lại, kích thước bằng chứng trung bình của chúng tôi trong trường hợp như vậy chỉ là 593 byte (một cải tiến của 2,5 lần), biện minh cho quyết định của chúng tôi để đặt tất cả các giá trị trong lá.



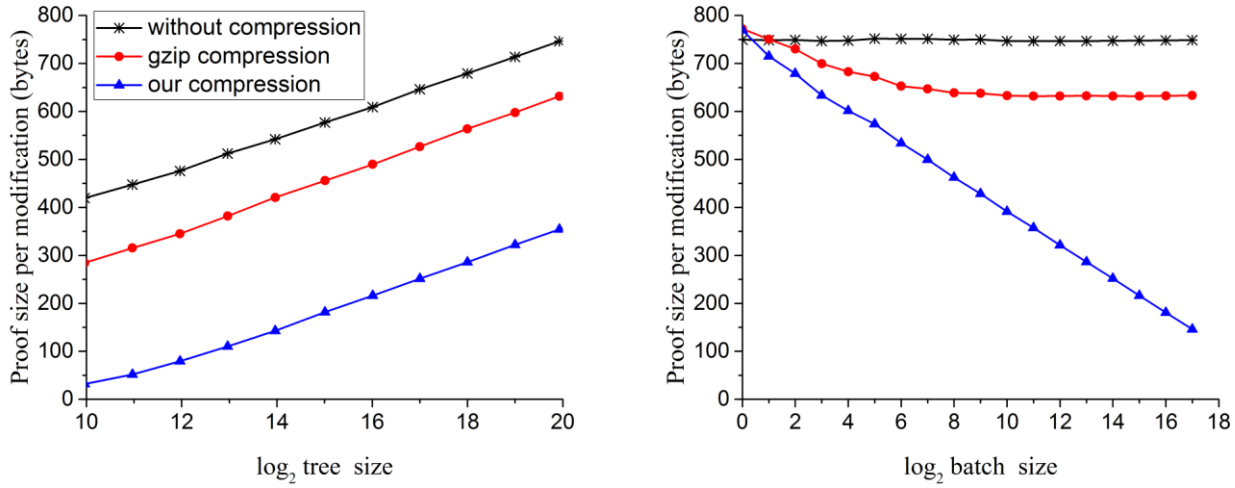
Cuối cùng, Ethereum triển khai cây Merkle Patricia Trie [Woo14, Phụ lục D] trong một mô hình tương tự như mô hình ba bên (vì nó không triển khai bằng chứng cho những thay đổi đối với bộ ba). Trong các thử nghiệm của chúng tôi (đã sử dụng mã Code từ [Tea16, trie/proof.go] để tạo bằng chứng cho cùng độ dài tham số như của chúng tôi) bằng cách sử dụng *n* từ 2000 đến 1.000.000, bằng chứng tra cứu của Ethereum luôn dài hơn 3 lần so với những thử nghiệm của chúng tôi dựa trên AVL+. Việc triển khai cây Merkle AVL+ của Tendermint [Kwo16] không có điều khoản nào để chứng minh không có khóa (cũng như chứng minh bất kỳ sửa đổi nào, vì nó nằm trong mô hình ba bên), nhưng dường như có độ dài bằng chứng gần giống như của chúng tôi khi điều chỉnh cho Hash và độ dài khóa.

Độ dài bằng chứng cho nhiều hoạt động. Việc nén cùng lúc các bằng chứng cho một lô B thao tác cùng một lúc (sử dụng Quan sát 6 trong Phần 3) giảm độ dài bằng chứng cho mỗi thao tác khoảng $36 \cdot \log_2 B$ byte. Cải tiến này lớn hơn đáng kể so với những gì chúng tôi có thể đạt được bằng cách ghép nối các bằng chứng riêng lẻ rồi áp dụng Gzip [GA], theo thử nghiệm, chưa bao giờ vượt quá 150 byte với mọi kích thước lô. Những cải tiến được báo cáo trong phần này và trong Hình 1 dành cho các khóa ngẫu nhiên thống nhất;

những sai lệch trong phân phối khóa chỉ có thể giúp ích cho quá trình nén của chúng tôi, vì chúng dẫn đến nhiều sự trùng lặp hơn giữa các đường dẫn dạng cây được sử dụng trong quá trình hoạt động.

Ví dụ: đối với n = 1.000.000, bằng chứng kết hợp cho 1000 lần cập nhật và 1000 lần chèn chỉ là 358 byte cho mỗi thao tác. Nếu một giao dịch trong một Block sửa đổi hai tài khoản và có 1.000.000 tài khoản và 1000 giao dịch trong Block (con số này là thực tế, xem [tbp]), thì chúng tôi có thể nhận được bằng chứng về 716 byte cho mỗi giao dịch còn lại ở mức bảo mật 128 bit. Nếu một số tài khoản hoạt động nhiều hơn và tham gia vào nhiều giao dịch, thì không gian trên mỗi giao dịch thậm chí còn ít hơn, vì thuật toán nén của chúng tôi hoạt động tốt hơn khi các hoạt động chia sẻ đường dẫn.

Chúng tôi có thể so sánh kết quả của mình với kết quả được báo cáo trong [MHKS14, Hình 13d] của Miller và cộng sự, người báo cáo kết quả của việc gộp lại với nhau (sử dụng bộ đệm "tạm thời" để loại bỏ một số nhãn và Gzip để nén luồng) B = 100.000 bằng chứng cho các hoạt động tra cứu trên cây đỏ đen, cây có kích thước n từ 2^4 đến 2^{21}. Đối với các phạm vi tham số này, bằng chứng của chúng tôi ngắn hơn ít nhất 2,4 lần, mặc dù chúng tôi sử dụng Hash dài hơn 1,6 lần, cũng như các khóa và giá trị dài hơn. Ví dụ: đối với $n = 2^{21}$, bằng chứng của chúng tôi chiếm 199 byte cho mỗi thao tác so với 478 byte của [MHKS14]. Bằng chứng cho các lần chèn thậm chí còn dài hơn trong [MHKS14], trong khi trong công việc của chúng tôi, chúng giống như đối với tra cứu. Một lần nữa, chúng tôi nhấn mạnh rằng công việc của Miller và cộng sự có ưu điểm là hỗ trợ các cấu trúc dữ liệu chung.



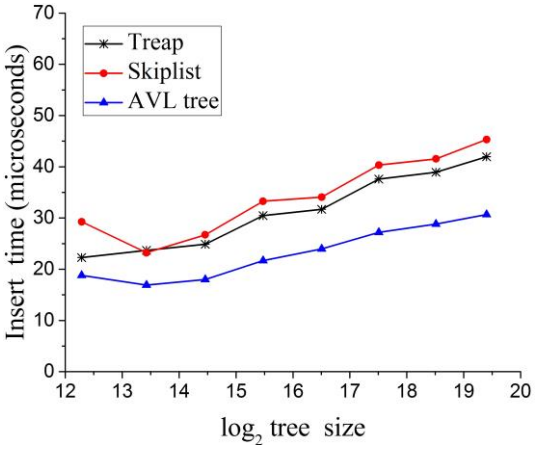
Hình 1: Bên trái: kích thước bằng chứng cho mỗi sửa đổi đối với B = 2000, như một hàm của kích thước cây bắt đầu n. Bên phải: kích thước bằng chứng cho mỗi lần sửa đổi đối với cây có n = 1.000.000 khóa, như một hàm của kích thước lô B. Trong cả hai trường hợp, một nửa số sửa đổi là chèn các cặp (khóa, giá trị) mới và một nửa là thay đổi giá trị cho các khóa hiện có.

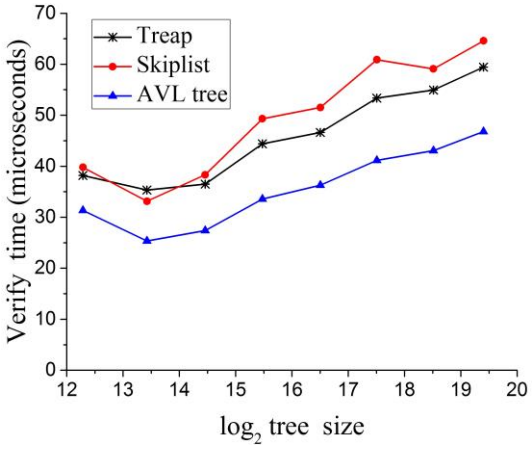
Thời gian chạy của Trình chứng minh và Trình xác minh. Các điểm chuẩn dưới đây được chạy trên máy Linux CPU Intel(R) Core(TM) i7-5820K @ 3,30GHz với RAM 8GB chạy ở chế độ 64 bit và chỉ sử dụng một Core. Chúng tôi đã sử dụng Java 8.0.51 và biên dịch mã Scala của mình bằng scalac 2.11.8. Việc triển khai Java của hàm Hash Blake2b được lấy từ trang web chính thức của Blake https://blake2.net/. Thời gian xác minh trung bình để chèn một khóa ngẫu nhiên vào cây AVL+ của chúng tôi với 1.000.000 khóa ngẫu nhiên là 31 μs, trong khi thời gian xác minh trung bình cho cùng một thao tác là 47 μs.

Rất khó để so sánh thời gian chạy giữa các triển khai do các biến thể trong môi trường phần cứng, ngôn ngữ lập trình được sử dụng, v.v. Tuy nhiên, lưu ý rằng bất kể các biến đó là gì, thời gian chạy của trình chứng minh và trình xác minh có tương quan chặt chẽ với độ dài đường dẫn *k*: trình chứng minh thực hiện *k* phép so sánh khóa (để tìm vị trí chèn) và tính toán *k* +1 giá trị Hash (để lấy nhãn của hai Node mới và *k* - 1 Node hiện tại có nhãn thay đổi), trong khi trình xác minh thực hiện hai phép so sánh (với khóa của hai lá lân cận) và tính toán 2*k* + 1 giá trị Hash (*k* để xác minh bằng chứng và *k* + 1 để tính toán thông báo mới). Việc xoay cây không thay đổi những con số này.

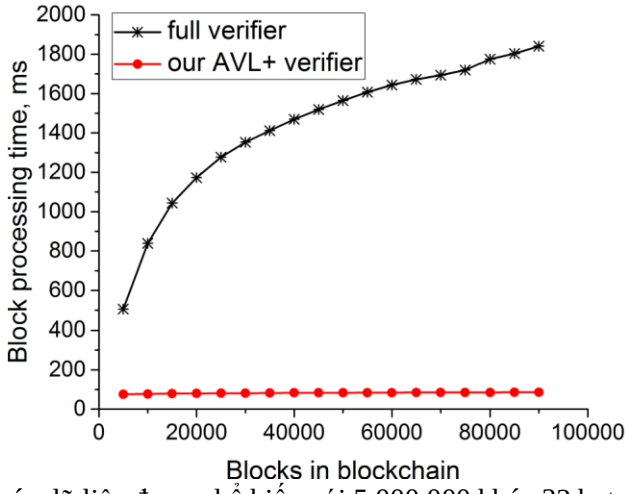
Do đó, chúng tôi mong đợi các cây AVL+ của chúng tôi hoạt động nhanh hơn khoảng 1,4 lần so với Skiplist, đó thực sự là những gì điểm chuẩn của chúng tôi cho thấy.

Khi chúng tôi gộp nhiều giao dịch lại với nhau, thời gian của trình chứng minh và trình xác minh sẽ cải thiện một chút khi kích thước lô tăng lên, đặc biệt là do nhãn của các Node không cần được tính toán cho đến khi toàn bộ lô được xử lý và do đó, nhãn của một số Node (những Node được tạo và sau đó được thay thế) không bao giờ được tính toán.





Chứng minh và xác minh Blockchain mô phỏng. Chúng tôi đã sử dụng máy chủ (máy Linux CPU Intel(R) Core(TM) i7-5820K @ 3,60GHz với 64GB RAM và bộ lưu trữ SSD) để mô phỏng hai phương pháp xác minh số dư tài khoản khác nhau: chỉ cần duy trì đầy đủ trên đĩa (SSD) cấu trúc dữ liệu của các cặp (khóa, giá trị) (tương tự như quy trình mà một "trình xác minh đầy đủ" truyền thống sẽ thực hiện) so với việc chỉ duy trì bản tóm tắt cấu trúc dữ liệu này và xác minh bằng chứng cho các hoạt động của cấu trúc dữ liệu, sử dụng rất ít RAM và không có trên ổ đĩa lưu trữ (tương tự như quy trình mà một "trình xác minh nhẹ" sẽ thực hiện khi các trình chứng minh sử dụng cây AVL+ của chúng tôi). Cấu trúc dữ liệu được phổ biến với 5.000.000 khóa 32 byte ngẫu nhiên (với các giá trị 8 byte) khi bắt đầu. Các Block mô phỏng của chúng tôi chứa 1500 bản cập nhật giá trị cho các khóa hiện có được chọn ngẫu nhiên và 500 lần chèn các khóa ngẫu nhiên mới. Chúng tôi đã



chạy mô phỏng cho 90000 Block (do đó kết thúc bằng cấu trúc dữ liệu gồm 50.000.000 khóa, tương tự như kích thước đặt Bitcoin UTXO [Lop] tại thời điểm viết).

Trình xác minh đầy đủ và nhẹ đều bị giới hạn ở 1GB RAM. Vì máy thực tế có 64GB RAM nên để ngăn hệ điều hành lưu toàn bộ cấu trúc dữ liệu trên ổ đĩa vào bộ nhớ đệm, chúng tôi đã mô phỏng một máy có RAM giới hạn bằng cách vô hiệu hóa toàn bộ bộ đệm ổ đĩa cấp hệ điều hành của trình xác minh cứ sau mỗi 10 giây. Chúng tôi chỉ đo thời gian xử lý cấu trúc dữ liệu và loại trừ thời gian để đọc Block từ mạng hoặc ổ đĩa để xác minh chữ ký trên các giao dịch, v.v. trình xác minh nhẹ duy trì ở mức khoảng 85 mili giây mỗi Block, giúp cấu trúc dữ liệu đã xác thực có lợi thế về tốc độ gấp 20 lần khi kích thước lớn hơn.

Để đảm bảo việc tạo bằng chứng là khả thi với một cỗ máy mạnh mẽ, chúng tôi đã chạy trình chứng minh của mình, nhưng cho phép nó sử dụng tối đa 48GB RAM. Trình chứng minh duy trì ở khoảng 70 mili giây mỗi Block, đây là một phần nhỏ trong tổng chi phí của một Node đầy đủ. Ví dụ: chi phí để xác minh 1000 chữ ký giao dịch, chỉ là một trong nhiều việc mà Node đầy đủ phải thực hiện để đưa các giao dịch vào một Block, là 280 mili giây trên cùng một máy (sử dụng sơ đồ chữ ký Ed25519 [BDL+12]). Kích thước bằng chứng thay đổi từ 0,8 đến 1 MB mỗi Block (nghĩa là 423-542 byte cho mỗi thao tác cấu trúc dữ liệu).

5 Phần kết luận

Chúng tôi đã chứng minh cải thiện hiệu suất đáng kể đầu tiên trong cấu trúc dữ liệu được xác thực hai bên kể từ [PT07] và cấu trúc dữ liệu được xác thực ba bên kể từ [CW11]. Chúng tôi đã làm như vậy bằng cách chỉ ra rằng Skiplist chỉ đơn giản là một trường hợp đặc biệt của cách tiếp cận cây tìm kiếm nhị phân cân bằng tổng quát hơn; tìm một cây tìm kiếm nhị phân tốt hơn để sử dụng; và phát triển một thuật toán để tập hợp các bằng chứng cho nhiều hoạt động. Chúng tôi cũng đã chứng minh có thể sử dụng cấu trúc dữ liệu được xác thực hai bên của chúng tôi để cải thiện đáng kể việc xác minh Blockchain bằng Node nhẹ mà không gây thêm gánh nặng cho Node đầy đủ, cung cấp minh chứng đầu tiên như vậy trong bối cảnh Crypto.

6 Sự ghi nhận

Chúng tôi cảm ơn Andrew Miller vì những giải thích hữu ích và chi tiết về công việc của anh ấy [MHKS14], vì đã chạy mã Code của anh ấy để lấy dữ liệu so sánh cho chúng tôi và vì những nhận xét về bản thảo của chúng tôi. Chúng tôi cảm ơn Peter Todd và Pieter Wuille vì những cuộc thảo luận hấp dẫn.

Tài liệu tham khảo

[AGT01] Aris Anagnostopoulos, Michael T. Goodrich và Roberto Tamassia. Từ điển được xác thực liên tục và các ứng dụng của chúng. Trong George I. Davida và Yair Frankel, biên tập viên, *Bảo mật thông tin, Hội nghị quốc tế lần thứ 4, ISC 2001, Malaga, Tây Ban Nha, ngày 1-3 tháng 10 năm 2001, Kỷ yếu*, tập 2200 của *Bài giảng Khoa học Máy tính*, trang 379–393. Springer, 2001. Có tại http://aris.me/pubs/pad.pdf.

[ANWOW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-OHearn và Christian Winnerlein. BLAKE2: đơn giản hơn, nhỏ hơn, nhanh như MD5. Trong *Hội nghị Quốc tế về Mật mã Ứng dụng và An ninh Mạng*, trang 119–135. Mùa xuân, 2013.

[AVL62] Adel'son-Vel'skii và Landis. Một thuật toán để tổ chức thông tin. *Dokladi Akademia Nauk SSSR* , 146(2), 1962. Bản dịch tiếng Anh bằng Toán học Liên Xô. Doklady 3, 1962, 1259–1263.

[BDL + 12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, và Bo-Yin Yang. Chữ ký bảo mật cao tốc độ cao. *J. Kỹ thuật mật mã* , 2(2):77–89, 2012.
Có sẵn tại https://ed25519.cr.yp.to/.

[BEG + 91] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, và Moni Naor. Kiểm tra tính chính xác của ký ức. Trong *Hội nghị chuyên đề thường niên lần thứ 32 về nền tảng của khoa học máy tính, San Juan, Puerto Rico, 1-4 tháng 10 năm 1991* , trang 90–99. IEEE Computer Society, 1991. Sau này có tên là [BEG + 94], có tại http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.2991.

[BEG + 94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan và Moni Naor. Kiểm tra tính chính xác của ký ức. *Algorithmica*, 12(2/3):225–244, 1994. Có tại http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.2991.

[BGM16] Iddo Bentov, Ariel Gabizon, và Alex Mizrahi. Crypto không có bằng chứng công việc. Trong Jeremy Clark, Sarah Meiklejohn, Peter YA Ryan, Dan S. Wallach, Michael Brenner và Kurt Rohloff, biên tập viên, *Mật mã tài chính và bảo mật dữ liệu - Hội thảo quốc tế FC 2016, BITCOIN, VOTING và WAHC, Christ Church, Barbados, ngày 26 tháng 2 năm 2016, Revised Selected Papers*, tập 9604 của *Bài giảng Khoa học Máy tính*, trang 142–157. Springer, 2016. Có tại http://arxiv.org/abs/1406.5694.

[BGV11] Siavosh Benabbas, Rosario Gennaro và Yevgeniy Vahlis. Ủy quyền tính toán có thể kiểm chứng trên các bộ dữ liệu lớn. Trong Phillip Rogaway, biên tập viên, *Những tiến bộ trong Mật mã học - CRYPTO 2011 - Hội nghị Mật mã học Thường niên lần thứ 31, Santa Barbara, CA, Hoa Kỳ, ngày 14-18 tháng 8 năm 2011. Kỷ yếu, tập 6841* của *Bài giảng Khoa học Máy tính*, trang 111–131. Springer, 2011. Có tại http://eprint.iacr.org/2011/132.

[BP07] Giuseppe Di Battista và Bernardo Palazzi. Các bảng quan hệ được xác thực và Skiplist được xác thực. Trong Steve Barker và Gail-Joon Ahn, biên tập viên, *Data and Applications Security XXI, 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Redondo Beach, CA, USA, 8-11 tháng 7 năm 2007, Kỷ yếu , tập 4602 của* Bài *giảng Notes in Computer Science*, trang 31–46. Springer, 2007. Có tại http://www.ece.umd.e du/~cpap/published/alex-ber-cpap-rt-08b.pdf .

[But16] Vitalik Buterin. Tấn công thư rác giao dịch: Các bước tiếp theo, 2016.
https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/.

[CDE+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi và Emin Gun. Mở rộng quy mô Blockchain phi tập trung. Trong *Proc. Hội thảo lần thứ 3 về Nghiên cứu Bitcoin và Blockchain*, 2016.

[CF13] Dario Catalano và Dario Fiore. Cam kết vectơ và ứng dụng của chúng. Trong Kaoru Kurosawa và Goichiro Hanaoka, biên tập viên, *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, 26 tháng 2 - 1 tháng 3, 2013. Kỷ yếu , tập 7778 của Bài giảng Ghi* chú *trong Khoa học Máy tính*, trang 55–72. Springer, 2013. Có tại http://eprint.iacr.org/2011/495.

[CLH+15] Xiaofeng Chen, Jin Li, Xinyi Huang, Jianfeng Ma, và Wenjing Lou. Cơ sở dữ liệu mới có thể kiểm chứng công khai với các bản cập nhật hiệu quả. *IEEE Trans. Dependable Sec. Comput.*, 12(5):546–556, 2015.

[CLW+16] Xiaofeng Chen, Jin Li, Jian Weng, Jianfeng Ma, và Wenjing Lou. Tính toán có thể kiểm chứng trên cơ sở dữ liệu lớn với các bản cập nhật gia tăng. *IEEE Trans. Máy tính*, 65(10):3184–3195, 2016.

[cod] Triển khai các cấu trúc dữ liệu được xác thực trong Scorex. https://github.com/input-output-hk/scrypto/.

[CW11] Scott A. Crosby và Dan S. Wallach. Từ điển xác thực: Chi phí và sự đánh đổi trong thế giới thực. *ACM Trans. Inf. Syst. Secur.*, 14(2):17, 2011. Có tại http://tamperevident.cs.rice.edu/Storage.html .

[DJ07] Brian C. Dean và Zachary H. Jones. Khám phá tính hai mặt giữa Skiplist và cây tìm kiếm nhị phân. Trong David John và Sandria N. Kerr, biên tập viên, *Kỷ yếu của Hội nghị Khu vực Đông Nam thường niên lần thứ 45, 2007, Winston-Salem, Bắc Carolina, Hoa Kỳ, ngày 23-24 tháng 3 năm 2007,* trang 395–399. ACM, 2007. Có tại https://people.cs.clemson.edu/~bcdean/skip_bst.pdf.

[DW13] Christian Decker và Roger Wattenhofer. Truyền bá thông tin trong mạng bitcoin. Trong *Kỷ yếu IEEE P2P 2013*, trang 1–10. IEEE, 2013.

[EK13] Mohammad Etemad và Alptekin Kupcu. Gia công cơ sở dữ liệu với cấu trúc dữ liệu được xác thực theo cấp bậc. In Hyang-Sook Lee and Dong-Guk Han, editors, *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers*, volume 8565 of *Lesson Notes in Computer Science*, trang 381–399. Springer, 2013. Có tại http://eprint.iacr .org/2015/351.

[GA] Jean-loup Gailly và Mark Adler. gzip. http://www.gzip.org/.

[GPT07] Michael T. Goodrich, Charalampos Papamanthou và Roberto Tamassia. Về chi phí kiên trì và xác thực trong Skiplist. Trong Camil Demetrescu, biên tập viên, *Experiment Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings* , volume 4525 của *Bài giảng Khoa học Máy tính*, trang 94–107. Springer, 2007. Có tại http://cs.brown.edu/cgc/stms/papers/pers-auth.pdf .

[GPTT08] Michael T. Goodrich, Charalampos Papamanthou, Roberto Tamassia và Nikos Triandopoulos. Athos: Xác thực hiệu quả các hệ thống tệp thuê ngoài. Trong TzongChen Wu, Chin-Laung Lei, Vincent Rijmen, và Der-Tsai Lee, biên tập viên, *Bảo mật thông tin, Hội nghị quốc tế lần thứ 11, ISC 2008, Đài Bắc, Đài Loan, tháng 9 năm 1518, 2008. Kỷ yếu*, tập 5222 của *Bài giảng Khoa học Máy tính*, trang 80–96. Springer, 2008. Có tại http://www.ece.umd.edu/~cpap/published/mtg-cpap-rt-nikos-08.pdf.

[GS78] Leonidas J. Guibas và Robert Sedgewick. Một khung lưỡng sắc cho cây cân bằng. Trong *Hội nghị chuyên đề thường niên lần thứ 19 về nền tảng của khoa học máy tính, Ann Arbor, Michigan, USA, 16-18 tháng 10 năm 1978*, trang 8–21. IEEE Computer Society, 1978. Có sẵn từ http://professor.ufabc.edu.br/~jesus.mena/courses/mc3305-2q-2015/AED2-13-redblack-paper.pdf.

[GSTW03] Michael T. Goodrich, Michael Shin, Roberto Tamassia và William H. Winsborough. Từ điển được xác thực cho thông tin đăng nhập thuộc tính mới. Ở Paddy Nixon và Sotirios Terzis, biên tập viên, *Trust Management, First International Conference, iTrust 2003, Heraklion, Crete, Hy Lạp, ngày 28-30 tháng 5 năm 2002, Kỷ yếu* , tập 2692 của *Bài giảng Khoa học Máy tính*, trang 332–347. Springer, 2003. Có tại http://cs.brown.edu/cgc/stms/papers/itrust2003.pdf .

[GT00] MT Goodrich và R. Tamassia. Từ điển được xác thực hiệu quả với Skiplist và Hash có tính giao hoán. Báo cáo Kỹ thuật, Viện Bảo mật Thông tin Johns Hopkins; có tại http://cs.brown.edu/cgc/stms/papers/hashskip.pdf , 2000.

[GTS01] MT Goodrich, R. Tamassia và A. Schwerin. Thực hiện một từ điển được xác thực với Skiplist và Hash có tính giao hoán. Có sẵn tại http://cs.brown. edu/cgc/stms/papers/discex2001.pdf; cũng được trình bày trong Proc. DARPA Information Survivability Conference & Exposition II (DISCEX II), 2001.

[HPPT08] Alexander Heitzmann, Bernardo Palazzi, Charalampos Papamanthou và Roberto Tamassia. Kiểm tra tính toàn vẹn hiệu quả của lưu trữ mạng không đáng tin cậy. Trong Yongdae Kim và William Yurcik, biên tập viên, *Kỷ yếu Hội thảo ACM 2008 về Bảo mật Lưu trữ và Khả năng sống sót, StorageSS 2008, Alexandria, VA, Hoa Kỳ, ngày 31 tháng 10 năm 2008, trang* 43–54. ACM, 2008. Có tại http://www.ece.umd.edu/~cpap/publish/alex-ber-cpap-rt-08b.pdf.

[KKR + 16] Aggelos Kiayias, Ioannis Konstantinou, Alexander Russell, Bernardo David và Roman Oliynykov. Một giao thức Blockchain bằng chứng cổ phần an toàn đã được chứng minh. Cryptology ePrint Archive, Báo cáo 2016/889, 2016. http://eprint.iacr.org/2016/889.

[Knu98] Donald Knuth. *Nghệ thuật lập trình máy tính: Tập 3: Sắp xếp và tìm kiếm*. Addison-Wesley, tái bản lần thứ 2, 1998.

[Kwo16] Jae Kwon. Tendermint go-merkle, 2016. https://github.com/tendermint/go-merkle.

[Lop] Jameson Lopp. Đầu ra giao dịch chưa chi tiêu trong Bitcoin. http://statoshi.info/dashboard/db/unspent-transaction-output-set, truy cập ngày 7 tháng 11 năm 2016.

[Mer89] Ralph C. Merkle. Một chữ ký số được chứng nhận. Trong Gilles Brassard, biên tập viên, *Những tiến bộ trong Mật mã học - CRYPTO '89, Hội nghị Mật mã học Quốc tế Thường niên lần thứ 9, Santa Barbara, California, Hoa Kỳ, ngày 20-24 tháng 8 năm 1989, Kỷ yếu, tập 435* của *Bài giảng Khoa học Máy tính*, trang 218–238. Springer, 1989. Có tại http: //www.merkle.com/papers/Certified1979.pdf .

[MHKS14] Andrew Miller, Michael Hicks, Jonathan Katz và Elaine Shi. Cấu trúc dữ liệu được xác thực, tổng quan. Trong Suresh Jagannathan và Peter Sewell, biên tập viên, *Hội nghị chuyên đề ACM SIGPLAN-SIGACT thường niên lần thứ 41 về Nguyên tắc ngôn ngữ lập trình, POPL '14, San Diego, CA, Hoa Kỳ, ngày 20-21 tháng 1 năm 2014, trang 411–424*. ACM, 2014. Trang dự án và phiên bản đầy đủ tại http://amiller.github.io/lambda-auth/paper.html.

[Mil12] Andrew Miller. Lưu trữ UTXO trong cây Merkle cân bằng (các Node không tin cậy với O(1)-storage), 2012. https://bitcointalk.org/index.php?topic=101734. msg1117428.

[Mil16] Andrew Miller. Truyền thông cá nhân, 2016.

[BQĐ+04] Charles U. Martel, Glen Nuckolls, Premkumar T. Devanbu, Michael Gertz, April Kwong và Stuart G. Stubblebine. Một mô hình chung cho các cấu trúc dữ liệu được xác thực. *Algorithmica*, 39(1):21–41, 2004. Có tại http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.75.3658 .

[MR98] Conrado Martinez và Salvador Roura. Cây tìm kiếm nhị phân ngẫu nhiên. *J. ACM* , 45(2):288–323, 1998. Nhãn có sẵn tại http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.243.

[MWMS16] Meixia Miao, Jianfeng Wang, Jianfeng Ma và Willy Susilo. Cơ sở dữ liệu có thể kiểm chứng công khai với các thao tác chèn/xóa hiệu quả. *Tạp chí Khoa học Máy tính và Hệ thống*, 2016. Có sẵn trực tuyến tại http://dx.doi.org/10.1016/j.jcss.2016.07.005. Để xuất hiện trong bản in.

[Nak08] Satoshi Nakamoto. Bitcoin: Một hệ thống tiền mặt điện tử ngang hàng. https://bitcoin.org/bitcoin.pdf, 2008.

[NN00] Moni Naor và Kobbi Nissim. Thu hồi chứng chỉ và cập nhật chứng chỉ. *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000. Có tại http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.7072.

[nxt] Crypto Nxt. https://nxt.org/.

[Pap11] Charalampos Papamanthou. *Mật mã cho hiệu quả: Hướng đi mới trong cấu trúc dữ liệu được xác thực*. Luận án tiến sĩ, Đại học Brown, 2011. Có tại http://www.ece.umd.edu/~cpap/published/theses/cpap-phd.pdf.

[Par15] Luca Parker. Sự sụt giảm trong các Node đầy đủ của bitcoin, năm 2015. http://bravenewcoin.com/ news/the-decline-in-bitcoins-full-nodes/.

[Pfa02] Ben Pfaff. GNU libavl 2.0.2, 2002. Có tại http://adtinfo.org/libavl.html/index.html.

[PT07] Charalampos Papamanthou và Roberto Tamassia. Các thuật toán hiệu quả về thời gian và không gian cho cấu trúc dữ liệu được xác thực của hai bên. Trong Sihan Qing, Hideki Imai, và Guilin Wang, biên tập viên, *An ninh Thông tin và Truyền thông, Hội nghị Quốc tế lần thứ 9, ICICS 2007, Trịnh Châu, Trung Quốc, ngày 12-15 tháng 12 năm 2007, Kỷ yếu, tập 4861* của *Bài giảng Khoa học Máy tính*, trang 1– 15. Mùa xuân, 2007. Có tại http://www.ece.umd.edu/~cpap/published/cpap-rt-07.pdf.

[PTT16] Charalampos Papamanthou, Roberto Tamassia và Nikos Triandopoulos. Các bảng Hash được xác thực dựa trên bộ tích lũy mật mã. *Thuật toán*, 74(2):664–712, 2016.

[Pug90] William Pugh. Skiplist: Một thay thế xác suất cho cây cân bằng. *Commu. ACM*, 33(6):668–676, 1990. Có tại http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.9072.

[Rog06] Phillip Rogaway. Chính thức hóa sự thiếu hiểu biết của con người. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 của *Bài giảng Khoa học máy tính*, trang 211–228. Mùa xuân, 2006.
Có tại https://eprint.iacr.org/2006/281.pdf.

[SA96] Raimund Seidel và Cecilia R. Aragon. Cây tìm kiếm ngẫu nhiên. *Algorithmica, 16(4/5):464–497*, 1996. Có tại https://faculty.washington.edu/aragon/pubs/rst96.pdf.

[ca] Ngôn ngữ lập trình Scala. http://www.scala-lang.org/.

[Sed08] Robert Sedgewick. Cây đỏ đen nghiêng trái, 2008. Có tại http://www.cs.Princeton.edu/~rs/talks/LLRB/LLRB.pdf .

[tbp] Giao dịch trên Block. https://blockchain.info/charts/n-transactions-per-block.

[Tea16] Nhóm Go Ethereum. Triển khai Golang chính thức của giao thức Ethereum, 2016. http://geth.ethereum.org/.

[Tod16] Peter Todd. Làm cho tăng trưởng bộ UTXO không liên quan với các cam kết TXO bị trì hoãn có độ trễ thấp, năm 2016. https://petertodd.org/2016/delayed-txo-commitments.

[Wei06] Mark Allen Weiss. *Cấu trúc dữ liệu và phân tích thuật toán trong Java (Ấn bản thứ hai)*. Pearsson, 2006.

[Whi15] Bill White. Một lý thuyết cho sổ cái Crypto nhẹ. Có tại http://qeditas.org/lightcrypto.pdf (xem thêm mã tại https://github.com/bitmyapp/ledgertheory), 2015.

[Wik13] Wiki về Bitcoin. CVE-2013-2293: Lỗ hổng DoS mới bằng cách buộc hoạt động đọc/tìm kiếm đĩa cứng liên tục, 2013. https://en.bitcoin.it/wiki/CVE-2013-2293.

[Woo14] Gavin Wood. Ethereum: Sổ cái giao dịch tổng quát phi tập trung an toàn. Có tại http://gavwood.com/Paper.pdf, 2014.

A Định nghĩa về bảo mật

Ký hiệu và chức năng. Giả sử rằng một cấu trúc dữ liệu trải qua một loạt các trạng thái S_0, S_1, ..., trong đó S_i là kết quả của việc áp dụng một số thao tác $\mathbf{op}_i$ cho trạng thái S_{i-1}. Trạng thái S_0 (ví dụ: một cây có một $-\infty$ Node gốc giám sát và bảo vệ) được mọi người biết đến. Trạng thái cấu trúc dữ liệu là *hợp lệ* nếu nó có thể được lấy từ S_0 bằng một chuỗi các hoạt động (ví dụ: cây AVL không cân bằng là không hợp lệ); chúng ta không bao giờ nên bắt đầu với cấu trúc dữ liệu không hợp lệ và các định nghĩa hàm và bảo mật của chúng ta không đảm bảo trong trường hợp như vậy. Nếu một thao tác không thay đổi trạng thái thì $S_i=S_{i-1}$. Ngoài khả năng thay đổi trạng thái, thao tác cũng có thể trả về giá trị $\mathbf{ret}_i$. Giả định rằng sự thay đổi trong trạng thái và giá trị trả về là xác định; cấu trúc dữ liệu ngẫu nhiên được mô hình hóa bằng cách chỉ định rõ ràng tính ngẫu nhiên được sử dụng như một phần của **op**. Có một hàm xác định hiệu quả D nhận trạng thái S và tính toán thông báo $D(S)$; đặt $D_0 = D(S_0)$.

Trình chứng minh trung thực có S_{i-1} (cộng với thông tin khác, ví dụ: nhãn của các Node cần thiết để xác thực cấu trúc dữ liệu, thông tin này chúng tôi không đưa vào một cách rõ ràng để tránh ký hiệu quá tải) và, ngoài việc thực hiện thao tác $\mathbf{op}_i$, đưa ra một bằng chứng π_i, chuyển đến trình xác minh. Đặt thuật toán của trình chứng minh được ký hiệu là $\mathrm{P}(S_{\mathrm{old}}, \mathbf{op}) \rightarrow (\pi, S_{\mathrm{new}}, \mathbf{ret})$ và thuật toán của trình xác minh được ký hiệu là $\mathrm{V}(D_{\mathrm{old}}, \mathbf{op}, \pi) \rightarrow$ (“accept”/“reject”, D’, **ret**’). Yêu cầu *về tính đầy đủ* (còn được gọi là tính chính xác) về mặt hình thức là đối với mọi trạng thái cấu trúc dữ liệu hợp lệ S_{old} và mọi thao tác **op**, nếu $D_{\mathrm{old}} = D(S_{\mathrm{old}})$, thuật toán $\mathrm{P}(S_{\mathrm{old}}, \mathbf{op})$ xuất ra $(\pi, S_{\mathrm{new}}, \mathbf{ret})$ sao cho $\mathrm{V}(D_{\mathrm{old}}, \mathbf{op}, \pi)$ chấp nhận và xuất ra $D' = D(S_{\mathrm{new}})$ và r**et**’ = **ret**.

Mục tiêu bảo mật chính: Tính đúng đắn. Yêu cầu bảo mật chính, được gọi là *tính đúng đắn*, là không kẻ tấn công có giới hạn tính toán nào có thể đưa ra bằng chứng π’ khiến trình xác minh chấp nhận và đưa ra kết quả D'_{new} hoặc **ret**’ không chính xác.

Bởi vì bảo mật của chúng tôi dựa trên tính bảo mật của Hash chống va chạm đối với hàm Hash cố định, nên chúng ta không thể nói về sự “không tồn tại” của một kẻ tấn công như vậy, bởi vì một khi hàm được cố định, một kẻ tấn công biết xung đột tồn tại theo nghĩa toán học (ngay cả khi không ai biết cách xây dựng). Thay vào đó, chúng ta nói về một thuật toán hiệu quả biến một đối thủ như vậy thành một xung đột Hash, theo kiểu [Rog06].

Về mặt hình thức, yêu cầu bảo mật được xây dựng dưới dạng Hash chống va chạm H, được sử dụng trong các thuật toán của trình chứng minh và trình xác minh và thuật toán D. Gọi bộ ba $(S, \mathbf{op}, \pi^*)$ *độc hại* nếu S là trạng thái cấu trúc dữ liệu hợp lệ nhưng $\mathrm{V}(D(S), \mathbf{op}, \pi^*)$ xuất ra (“accept”, D'_{new}, **ret**’) và $D'_{\mathrm{new}} \neq D(S_{\mathrm{new}})$ hoặc **ret**’ $\neq$ **ret**. Yêu cầu bảo mật là tồn tại một thuật toán hiệu quả R, với bất kỳ bộ ba độc hại nào làm đầu vào, sẽ cho ra $x \neq y$ sao cho $H(x) = H(y)$.

Mục tiêu bảo mật phụ: Đảm bảo tính hiệu quả của Trình xác minh. Mục tiêu bảo mật phụ của chúng tôi để giảm khả năng tấn công từ chối dịch vụ đối với trình xác minh là trình xác minh hoạt động hiệu quả bất kể trình chứng minh trung thực hay ác ý làm việc gì. Cụ thể, chúng tôi yêu cầu đối với cấu trúc dữ liệu có n phần tử, thời gian chạy trình xác minh được đảm bảo $O(\log n)$ bất kể đầu vào là gì. Nghĩa là nếu S là một cấu trúc dữ liệu hợp lệ với n phần tử và $D(S)$ là bản tóm tắt của nó, thì $\mathrm{V}(D(S), \mathbf{op}, \pi^*)$ hoàn thành thao tác của nó trong thời gian $O(\log n)$ cho bất kỳ **op** và π^* nào (đặc biệt, điều này ngụ ý rằng V thậm chí sẽ không đọc được π^* quá dài và các bằng chứng trung thực π sẽ luôn có độ dài $O(\log n)$). Đối với lô B thao tác được chứng minh cùng nhau, thời gian chạy trình xác minh phải được đảm bảo $O(B \log(n + B))$.

B Bằng chứng bảo mật

Thuật toán của chúng tôi. Cấu trúc chính xác của π và thuật toán xác minh chính xác không quan trọng đối với bằng chứng bảo mật. Dưới đây là những tính năng nổi bật mà chúng tôi sử dụng để chứng minh tính bảo mật.

V tạo một phần cây T bằng cách sử dụng thông tin trong **op** và trong π. Cây này bắt đầu từ cùng một gốc, nhưng một số nhánh kết thúc sớm hơn các nhánh tương ứng trong S. Mọi Node trong T có thể là Node *chỉ có nhãn*, không chứa Node con và không có thông tin nào khác ngoài nhãn (Node như vậy kết thúc một nhánh) hoặc Node *nội dung*, chứa tất cả các trường giống như Node tương ứng trong S (bao gồm 1 bit cho biết đó có phải là lá hay không), ngoại trừ việc bỏ qua khóa trong trường hợp Internal Node. Mọi Node nội dung trong T hoặc là một lá hoặc có cả hai Node con (do đó, mọi Node trong T đều có các Node cùng cấp độ của nó). Nhãn của các Node chỉ có nhãn được đọc từ bằng chứng, trong khi nhãn của các Node nội dung được V lấy thông qua ứng dụng của H (và không bao giờ được đọc từ bằng chứng). V kiểm tra xem nhãn gốc của T có khớp với nhãn gốc của S chứa trong $D(S)$ hay không.

V thực hiện thao tác tương tự trên T giống như P thực hiện trên S để tính toán **ret** và $D(S_{\text{new}})$, với sự khác biệt quan trọng sau: khi tìm kiếm khóa **key** yêu cầu so sánh giữa **key** và **t.key** cho một số Node **t** không có là lá trong T để quyết định đi sang trái hay sang phải, V không thực hiện so sánh mà đọc hướng trái hoặc phải từ bằng chứng. Sau đó, khi tìm kiếm đến một lá **f** (và mọi tìm kiếm như vậy phải đến một lá, nếu không thì V từ chối), V kiểm tra xem **f.key** $\leq$ **key** $<$ **f.nextKey** (đặc biệt, nếu **f.key** $<$ **key** thì V xác định rằng không tìm thấy khóa). V cũng từ chối nếu T không chứa đủ các Node nội dung để tính các giá trị **ret** và $D(S_{\text{new}})$.

Chúng tôi không đề cập đến tính đầy đủ ở đây, vì nó rất dễ nhận thấy.

Tính đúng đắn. Bây giờ chúng ta cần chỉ ra thuật toán tìm va chạm $R(S, \mathbf{op}, \pi^*)$. R sẽ chạy trình xác minh $\mathrm{V}(D(S), \mathbf{op}, \pi^*)$ để lấy cây xác minh một phần T^*. Chúng ta sẽ nói rằng Node $\mathbf{t}^\bullet$ trong cây xác minh T^* không khớp với Node **t** trong S nếu các Node ở cùng một vị trí (được xác định bởi đường đi từ gốc), nhưng một số thông tin trong $\mathbf{t}^\bullet$ mâu thuẫn với thông tin tương ứng trong **t**. (Lưu ý rằng Node xác minh có thể không có tất cả thông tin của Node chứng minh, nó có thể chỉ chứa nhãn hoặc không chứa khóa, nhưng riêng điều này không phải là lý do để gọi các Node là "không khớp". Thông tin hiện có trong cả hai Node phải khác nhau để các Node được coi là không khớp.)

Nếu $\mathbf{t}^\bullet$ tồn tại trong T^*, nhưng một Node **t** ở cùng vị trí hoặc không tồn tại trong S hoặc không khớp với $\mathbf{t}^\bullet$, thì R có thể dễ dàng tìm thấy xung đột Hash. Đầu tiên, tìm một $\mathbf{t}^\bullet$ cao nhất trong T^* mà điều này là đúng (phá vỡ các ràng buộc một cách tùy ý). Một Node ở cùng một vị trí phải tồn tại trong S vì lý do sau: nếu $\mathbf{t}^\bullet$ là gốc của T^*, thì một Node ở cùng một vị trí tồn tại trong S vì S không bao giờ rỗng, vì S có một Node giám sát và bảo vệ. Mặt khác, cha $\mathbf{p}^\bullet$ của $\mathbf{t}^\bullet$ là Node cao hơn và là Node nội dung và không có lá, có nghĩa là Node

p khớp với $\mathbf{p}^*$ phải tồn tại trong *S* và cũng phải là Node không có lá (vì các bit chỉ thị lá của **p** và $\mathbf{p}^*$ phải bằng nhau), và mọi Node không có lá trong *S* đều có 2 con.

Bây giờ xét $\mathbf{t}^*$ và **t**. Nếu nhãn của hai Node này không khớp nhau, thì chúng không phải là gốc (vì V kiểm tra xem nhãn của gốc của T^* có khớp với nhãn của gốc của *S* có trong bản tóm tắt $D(S)$). Do đó, nhãn của cha mẹ của chúng khớp nhau (vì $\mathbf{t}^*$ và **t** là cặp không khớp cao nhất), và do đó chúng tôi tìm thấy xung đột Hash trong tính toán của nhãn cha (hãy nhớ là cây xác minh T^* có các Node cùng cấp, và do đó, nhãn các Node cùng của $\mathbf{t}^*$ có sẵn cho *R* để tạo đầu vào cho *H*). Nếu nhãn của $\mathbf{t}^*$ và **t** trùng nhau thì một số nội dung khác không khớp; do đó, $\mathbf{t}^*$ là một Node nội dung và tất cả nội dung cần thiết để tính toán nhãn của nó nằm trong T^* và do đó có sẵn cho *R*. Do đó, *R* có thể tạo ra xung đột Hash trên nội dung của hai Node này.

Vẫn còn phải xem xét trường hợp khi mọi Node trong T^* có một Node tương ứng trong *S* và khớp với nó. Trong trường hợp này, chúng ta sẽ rút ra mâu thuẫn với mệnh đề **ret**' $\neq$ **ret** hoặc $D' \neq S_{new}$. Đặt **key** là khóa được chỉ định trong **op**. Để V chấp nhận, đường dẫn tìm kiếm được chỉ ra trong bằng chứng phải dẫn đến một lá $\mathbf{f}^*$ trong T^* với **f.key** $\leq$ **key** $<$ **f.nextKey**. Đặt **f** là lá phù hợp trong *S*. Vì *S* hợp lệ nên chỉ có một lá trong *S* sao cho **f.key** $\leq$ **key** $<$ **f.nextKey** và do đó, đường dẫn tìm kiếm của trình chứng minh trung thực sẽ dẫn đến **f**. Do đó, con đường tìm kiếm của V cũng giống như con đường tìm kiếm của trình chứng minh trung thực P. Tất cả các bước khác của V cũng giống như trên T^* như các bước tương ứng của P trung thực trên *S* theo thiết kế. Do đó, chúng thực hiện các bước giống nhau trên cùng một cây và V sẽ tính toán cùng một **ret** và $D(S_{new})$ như P.

Đảm bảo hiệu quả của Trình xác minh: Trường hợp hoạt động đơn lẻ. Các cây AVL+ của chúng tôi cũng đáp ứng mục tiêu bảo mật phụ về hiệu quả của trình xác minh được đảm bảo. Lý do cơ bản là trình xác minh có thể từ chối bất kỳ bằng chứng nào chứa nhiều hơn $O(\log n)$ Node cho mỗi thao tác. Bây giờ chúng tôi trình bày một dẫn xuất cẩn thận hơn, bao gồm các kiểm tra chính xác mà V phải thực hiện, dành cho những người quan tâm đến chi tiết.

Xác định chiều cao của một lá là 0 và chiều cao của một Internal Node là 1 cộng với chiều cao tối đa của hai con của nó. Chúng tôi thêm chiều cao gốc *h* vào thông báo của *S* (nó có thể được duy trì bởi cả trình chứng minh và trình xác minh, bởi vì cả hai bên đều có thể cho biết chiều cao thay đổi như thế nào sau khi áp dụng một thao tác). Trong cây AVL+ có *n* lá cộng với một lá giám sát, chiều cao này được đảm bảo tối đa là $1{,}4405\log_2(n+2)$ (xem [Knu98, trang 460] và quan sát thấy rằng số Internal Node bằng số lượng lá không có giám sát). Bằng chứng cho các lần chèn, tra cứu và cập nhật được phép chứa một đường dẫn duy nhất có độ dài lên tới *h* (theo độ dài đường dẫn, chúng tôi muốn nói đến số lượng không có lá trên đường dẫn): nghĩa là, tối đa *h* Internal Node, một lá, và tối đa *h* Node chỉ có nhãn.

Bằng chứng cho việc xóa được phép chứa hai đường dẫn có độ dài lên tới *h* cho mỗi đường (sẽ trùng nhau trong ít nhất một Node - gốc của cây). Ngoài ra, đối với một trong hai đường dẫn, các Node cách đường dẫn này một hoặc hai bước theo hướng ngoài đường dẫn được cho phép trong bằng chứng về việc xoay của cây. Chúng ta có thể nói chính xác hơn: tìm kiếm ngoài đường dẫn để thực hiện xoay sau khi xóa chỉ cần thiết khi cây con ngoài đường dẫn cao hơn cây con trên đường dẫn trước khi xóa, có nghĩa là chênh lệch độ cao giữa Node và Node con của nó bằng 2. Do đó, nếu đường dẫn từ gốc có độ cao *h* đến lá (có độ cao 0) có độ dài *s*, thì có tối đa $h - s$ Node mà tại đó có thể xảy ra việc xoay của cây; mỗi vòng xoay yêu cầu tối đa 2 Node ngoài đường dẫn. Do đó, tổng số Internal Node mà bằng chứng có thể chứa nhiều nhất là $2(h-s) + s + h - 1 = 3h - s - 1$. Cũng lưu ý rằng $s \geq h/2$ (nếu không thì chiều cao của một số Node trên đường dẫn khác với chiều cao của Node con hơn hai, điều này mâu thuẫn với điều kiện cân bằng AVL), và do đó tổng số Internal Node bằng chứng xóa nhiều nhất là $2{,}5h - 1$. Tổng số lá nhiều nhất là 2 và tổng số Node chỉ có nhãn nhiều nhất là $2{,}5h - 2$ (có thể dễ dàng nhận thấy bằng quy nạp về số Internal Node).

V có thể ngay lập tức từ chối bất kỳ π nào; vì thời gian chạy của V là tuyến tính theo π, nên chúng ta có thời gian chạy của V được đảm bảo là $O(\log n)$.

Đảm bảo hiệu quả của Trình xác minh: Trường hợp nhiều hoạt động. Khi chúng tôi nén các bằng chứng cho một loạt nhiều thao tác cùng nhau, đối số về kích thước bằng chứng tối đa sẽ phức tạp hơn một chút, bởi vì đối số ở trên về số lượng Node tối đa hiện phải được áp dụng cho mỗi thao tác, trong khi chỉ thao tác đầu tiên hoạt động trên S, trong khi những thao tác tiếp theo hoạt động trên những thao tác tiếp theo của S, cái này có thể tăng chiều cao do các Node mới được thêm vào và nhiều hoạt động tái cân bằng. May mắn thay, vì trình chứng minh chỉ giao tiếp với một cây con của S, nên các đường dẫn đến lá vẫn có độ dài tối đa là h (vì S không còn đường dẫn nào nữa). Tuy nhiên, chúng tôi không thể sử dụng cùng một giới hạn như trước đây về số Node ngoài đường dẫn cần thiết để cân bằng lại trong trường hợp xóa, bởi vì đối số đó dựa trên chiều cao của gốc tại thời điểm xóa đang diễn ra, có thể lớn hơn h. Do đó, chúng tôi tính toán chiều cao cây tối đa có thể xảy ra trong nhiều hoạt động.

Nếu S có chiều cao h, thì số lá của nó (bao gồm cả giám sát) nhiều nhất là 2^h; sau i lần chèn, con số này nhiều nhất là $2^h + i$, và do đó chiều cao nhiều nhất là $h_{\text{new}} \le 1{,}4405\log_2(2^h + i + 1) \le 1{,}4405\log_2(2\cdot\max(2^h, i + 1)) = 1{,}4405(1 + \max(h, \log_2(i + 1)))$. Do đó, số lượng Node ngoài đường dẫn cần thiết cho mỗi lần xóa nhiều nhất là $2(h_{\text{new}} - s)$, trong đó $s \ge h_{\text{new}}/2$ (và do đó $2(h_{\text{new}} - s) \le h_{\text{new}}$) theo cách lập luận tương tự như đối với bằng chứng thao tác đơn lẻ ở trên. Do đó, đối với $B > 0$ mà d là phép xóa, bằng chứng có thể chứa tối đa $(B + d)h + dh_{\text{new}}$ Internal Node (với $h_{\text{new}} = 1{,}4405(1 + \max(h, \log_2 B))$), không có nhiều Internal Node hơn các Node chỉ có nhãn và $B + d$ lá. (Ràng buộc này không chặt chẽ và có thể được cải thiện.) Một lần nữa, bất kỳ bằng chứng nào dài hơn sẽ bị V từ chối trước khi V hoàn thành việc tái tạo T. Sau khi T được xây dựng lại và thông báo của nó được xác minh, V sẽ mất thời gian logarit cho mỗi thao tác theo bảo đảm cây AVL, vì chiều cao của T nhiều nhất là h, vì T là cây con của S theo bằng chứng về tính hợp lý.

Người dịch: Nguyễn Văn Tú

Telegram: http://t.me/Tulibra

Link gốc: https://iohk.io/en/research/library/papers/improving-authenticated-dynamic-dictionaries-with-applications-to-cryptocurrencies/