**CryptDB: DBMS quan hệ được mã hóa thực tế**

Raluca Ada Popa, Nickolai Zeldovich và Hari Balakrishnan {raluca, nickolai, hari}@csail.mit.edu

# TRỪU TƯỢNG

*CryptDB* là một DBMS cung cấp quyền riêng tư có thể chứng minh và thực tế khi đối mặt với máy chủ cơ sở dữ liệu bị xâm nhập hoặc quản trị viên cơ sở dữ liệu tò mò. CryptDB hoạt động bằng cách thực hiện các truy vấn SQL trên dữ liệu được mã hóa. Cốt lõi của nó là ba ý tưởng mới: *chiến lược mã hóa nhận biết*  SQL ánh xạ các hoạt động SQL với các sơ đồ mã hóa, mã hóa dựa trên truy vấn có thể điều chỉnh *cho phép CryptDB điều chỉnh mức độ mã hóa của từng mục dữ liệu dựa trên truy vấn của người dùng và*  mã hóa hành tây để thay đổi hiệu quả mức mã hóa dữ liệu. CryptDB chỉ trao quyền cho máy chủ thực hiện các truy vấn mà người dùng yêu cầu và đạt được sự riêng tư tối đa với sự kết hợp của các truy vấn do người dùng đưa ra. Máy chủ cơ sở dữ liệu đánh giá đầy đủ các truy vấn về dữ liệu được mã hóa và gửi kết quả trở lại máy khách để giải mã cuối cùng; Máy khách không thực hiện bất kỳ xử lý truy vấn nào và các ứng dụng phía máy khách chạy không thay đổi. Đánh giá của chúng tôi cho thấy CryptDB có chi phí khiêm tốn: trên điểm chuẩn TPC-C trên Postgres, CryptDB giảm 27% thông lượng so với Postgres thông thường. Điều quan trọng là CryptDB không thay đổi nội dung của các DBMS hiện có: chúng tôi đã nhận ra việc triển khai CryptDB bằng cách sử dụng viết lại / mã hóa truy vấn phía máy khách, các hàm do người dùng xác định và các bảng phía máy chủ cho thông tin khóa công khai. Như vậy, CryptDB là di động; Việc chuyển CryptDB sang MySQL đòi hỏi phải thay đổi 86 dòng mã, chủ yếu ở lớp kết nối.

# GIỚI THIỆU

Đánh cắp dữ liệu cá nhân nhạy cảm là một vấn đề nghiêm trọng [37]. Hệ thống quản lý cơ sở dữ liệu (DBMS) là một mục tiêu đặc biệt hấp dẫn đối với những kẻ tấn công, bởi vì chúng thường chứa một lượng lớn thông tin cá nhân. Khi người dùng cá nhân hoặc doanh nghiệp lưu trữ dữ liệu nhạy cảm của họ trong DBMS ngày nay, họ phải tin tưởng rằng phần cứng và phần mềm máy chủ là không thể thỏa hiệp, rằng bản thân trung tâm dữ liệu được bảo vệ vật lý và cho rằng quản trị viên hệ thống và cơ sở dữ liệu (DBA) là đáng tin cậy. Mặt khác, kẻ thù có quyền truy cập vào bất kỳ con đường tấn công nào trong số này có thể làm tổn hại toàn bộ cơ sở dữ liệu, như đã được ghi nhận trong một số báo cáo được công bố về hành vi trộm cắp dữ liệu [37] (và có lẽ có nhiều thỏa hiệp chưa được công khai).

Những yêu cầu bảo mật nghiêm ngặt này cũng mâu thuẫn với chi phí-

|  |
| --- |
| Quyền tạo bản sao kỹ thuật số hoặc bản cứng của tất cả hoặc một phần tác phẩm này cho mục đích sử dụng cá nhân hoặc lớp học được cấp miễn phí với điều kiện là các bản sao không được tạo hoặc phân phối vì lợi nhuận hoặc lợi thế thương mại và các bản sao có thông báo này và trích dẫn đầy đủ trên trang đầu tiên. Để sao chép khác, để tái xuất bản, đăng trên máy chủ hoặc phân phối lại cho danh sách, cần có sự cho phép cụ thể trước và / hoặc một khoản phí. |

các biện pháp tiết kiệm như hợp nhất DBMS thuộc các đơn vị kinh doanh khác nhau thành một cơ sở hạ tầng CNTT chung trên toàn doanh nghiệp, di chuyển cơ sở dữ liệu vào đám mây công cộng hoặc thuê ngoài các tác vụ DBA. Trên thực tế, "thiếu tin tưởng" là mối quan tâm chính thường được trích dẫn về việc di chuyển dữ liệu trong các hệ thống cơ sở dữ liệu sang cơ sở hạ tầng đám mây hiệu quả hơn về chi phí. Hơn nữa, nhờ các vụ trộm cắp số nhận dạng an sinh xã hội, số thẻ tín dụng và thông tin cá nhân khác từ các cơ sở dữ liệu trực tuyến khác nhau, những lo ngại này cũng ngày càng được phản ánh trong luật: ví dụ, luật pháp gần đây yêu cầu tất cả các cơ sở dữ liệu chứa dữ liệu cá nhân về cư dân Massachusetts phải được mã hóa [28].

Bài viết này trình bày *CryptDB, một DBMS* quan hệ thực tế cung cấp các đảm bảo quyền riêng tư có thể chứng minh được mà không cần phải tin tưởng vào máy chủ DBMS hoặc các DBA duy trì và điều chỉnh DBMS. Trong CryptDB, các máy chủ DBMS chưa sửa đổi lưu trữ *tất cả*  dữ liệu ở định dạng được mã hóa và thực hiện các truy vấn SQL trên dữ liệu được mã hóa mà không có quyền truy cập vào các khóa giải mã. CryptDB hoạt động bằng cách chặn và viết lại  *tất cả các truy vấn SQL trong giao diện người dùng* để làm cho chúng thực thi trên dữ liệu được mã hóa, bằng cách mã hóa và giải mã tất cả dữ liệu, cũng như thay đổi một số toán tử truy vấn, trong khi vẫn giữ nguyên ngữ nghĩa tổng thể của truy vấn. Giao diện người dùng có quyền truy cập vào khóa mã hóa cho toàn bộ cơ sở dữ liệu. Bằng cách không cấp cho máy chủ DBMS quyền truy cập vào khóa giải mã, CryptDB làm giảm đáng kể các yêu cầu tin cậy cho máy chủ DBMS. Ví dụ: CryptDB có thể giảm bớt lo ngại về quyền riêng tư khi thuê ngoài cơ sở dữ liệu cho môi trường điện toán đám mây [47], chẳng hạn như AWS của Amazon, SQL Azure của Microsoft hoặc AppEngine của Google hoặc khi thuê ngoài công việc của DBA. CryptDB cũng có thể ngăn chặn vi phạm quyền riêng tư do quản trị viên tò mò [6] hoặc máy chủ DBMS bị xâm phạm.

Có ba thách thức đáng kể trong việc thiết kế và triển khai DBMS hoạt động trên dữ liệu được mã hóa. Đầu tiên nằm ở việc hỗ trợ một loạt các truy vấn SQL trên dữ liệu được mã hóa. Không giống như một kho dữ liệu được mã hóa đơn giản, DBMS phải  *thực hiện tính toán* trên dữ liệu được mã hóa để thực hiện các truy vấn SQL. Ví dụ: truy vấn SQL có thể yêu cầu mức lương trung bình của nhân viên, tên của nhân viên có mức lương lớn hơn 60.000 đô la hoặc danh sách nhân viên chia sẻ văn phòng với nhiều hơn hai đồng nghiệp. Chỉ cần mã hóa mỗi hàng trong cơ sở dữ liệu bằng một khóa duy nhất sẽ không cho phép máy chủ DBMS thực hiện các truy vấn SQL như vậy mà không cần truy cập vào khóa giải mã.

Thách thức thứ hai là xác định cẩn thận "quyền riêng tư" cho một DBMS không đáng tin cậy, cũng như đưa ra một thiết kế hệ thống chứng minh đạt được định nghĩa đó. Một mặt, ngay cả khi tất cả dữ liệu được lưu trữ trên máy chủ DBMS được mã hóa, máy chủ phải có khả năng thực hiện một số thao tác nhất định trên các hàng, chẳng hạn như tổng hợp, lựa chọn và nối. Mặt khác, một đối thủ xâm phạm máy chủ DBMS giờ đây có thể tìm hiểu thông tin về dữ liệu, chẳng hạn như mối quan hệ giữa các hàng khác nhau trong một bảng. Do đó, chúng ta cần xác định quyền riêng tư theo cách cân bằng giữa nhu cầu tính toán phía máy chủ với nhu cầu giảm thiểu thông tin được tiết lộ cho máy chủ.

Thách thức thứ ba nằm ở việc làm cho một DBMS được mã hóa trở nên thiết thực để sử dụng. Để cung cấp hiệu suất tốt, DBMS được mã hóa nên áp đặt chi phí hiệu suất tối thiểu trên máy chủ, nhưng đồng thời tránh giảm tải thực thi truy vấn SQL lên máy khách. Để làm cho DBMS được mã hóa dễ triển khai, một hệ thống lý tưởng cũng sẽ không yêu cầu  *thay đổi phần mềm máy chủ DBMS hiện có, để nó có thể tận dụng hơn bốn thập kỷ làm việc kỹ thuật và tối ưu hóa, chạy trên một loạt các máy chủ DBMS hàng hóa, cũng như* không thay đổi các ứng dụng khách*.*

Chúng tôi giải quyết những thách thức này trong thiết kế của CryptDB bằng cách sử dụng ba ý tưởng chính. Ý tưởng đầu tiên là một *chiến lược mã hóa nhận biết SQL*. Chúng tôi quan sát thấy rằng các truy vấn SQL bao gồm các toán tử nguyên thủy, đặc biệt là so sánh thứ tự, kiểm tra đẳng thức và bổ sung. Đối với hầu hết các toán tử này, chúng tôi đã tìm thấy các sơ đồ mã hóa hiện có trong đó hoạt động có thể được thực hiện trên bản mã mà không cần biết khóa giải mã. Một ngoại lệ là tham gia, mà không có nguyên thủy mật mã tồn tại; Trong trường hợp này, chúng tôi đã phát triển một cấu trúc mật mã mới để bảo vệ quyền riêng tư. Với những nguyên thủy này, chúng tôi mã hóa từng mục dữ liệu bằng các sơ đồ mã hóa cho phép máy chủ thực thi các toán tử SQL cần thiết trên dữ liệu đó. Cách tiếp cận này vừa hiệu quả vừa thực tế, vì phần lớn DBMS, bao gồm lập kế hoạch truy vấn, bố cục dữ liệu, phối hợp giao dịch và cấu trúc của chính các truy vấn, có thể giữ nguyên và chỉ các toán tử SQL riêng lẻ được sử dụng bởi truy vấn mới có thể cần thay đổi. Ý tưởng thứ hai là  *mã hóa dựa trên truy vấn có thể* điều chỉnh, trong đó CryptDB tự động điều chỉnh mức mã hóa cho từng mục dữ liệu trong thời gian chạy, để đạt được mức độ riêng tư tối đa cho các truy vấn của người dùng. Cụ thể, CryptDB ban đầu mã hóa tất cả dữ liệu với mức mã hóa mạnh nhất và khi ứng dụng đưa ra các truy vấn SQL, CryptDB điều chỉnh mức độ mã hóa trên máy chủ, để máy chủ có thể thực hiện các lớp tính toán cần thiết cho truy vấn SQL đó. Mô hình này tạo thành cơ sở cho định nghĩa quyền riêng tư của chúng tôi, đảm bảo quyền riêng tư tối đa với các lớp tính toán theo yêu cầu của các truy vấn được trình bày cho DBMS và tránh phải sửa đổi mã ứng dụng để khai báo mức mã hóa cần thiết trước thời hạn.

Ý tưởng thứ ba là thực hiện mã hóa dựa trên truy vấn có thể điều chỉnh bằng cách mã hóa từng mục dữ liệu trong một *củ hành* mã hóa, từ các hình thức mã hóa yếu hơn cho phép tính toán nhất định, đến các hình thức mã hóa mạnh hơn không tiết lộ thông tin, như thể hiện trong Hình 3. Cách tiếp cận này cho phép CryptDB điều chỉnh hiệu quả mức độ mã hóa trên máy chủ mà không cần phải mã hóa lại tất cả dữ liệu tại máy khách. Ví dụ, lớp ngoài cùng sử dụng mã hóa ngẫu nhiên, đảm bảo rằng máy chủ không thể tìm hiểu gì về dữ liệu, ngoài chiều dài của nó. Nếu người dùng phát hành truy vấn SQL có chứa WHERE id = 5, CryptDB sẽ gửi cho máy chủ một *khóa củ hành* để giải mã cột id đến mức mã hóa xác định, trong đó các bản rõ giống hệt nhau có bản mã giống hệt nhau.[[1]](#footnote-1) CryptDB sau đó gửi cho máy chủ một mã hóa xác định của hằng số 5, cho phép nó tính toán các hàng phù hợp bằng cách chỉ tiết lộ các mối quan hệ cần thiết giữa các mục dữ liệu *và không tiết lộ dữ liệu thực tế* hoặc các mối quan hệ khác giữa các mục dữ liệu không được sử dụng trong truy vấn này.

Theo hiểu biết của chúng tôi, CryptDB là hệ thống riêng đầu tiên hỗ trợ tất cả SQL tiêu chuẩn qua dữ liệu được mã hóa mà không yêu cầu bất kỳ xử lý truy vấn phía máy khách nào, sửa đổi cơ sở mã DBMS hiện có, thay đổi các ứng dụng cũ và giảm tải hầu như tất cả các xử lý truy vấn cho máy chủ. CryptDB hoạt động bằng cách viết lại các truy vấn SQL, lưu trữ dữ liệu được mã hóa trong các bảng thông thường và sử dụng hàm do người dùng SQL xác định (UDF) để thực hiện các hoạt động mật mã phía máy chủ. Chúng tôi đã triển khai một nguyên mẫu của CryptDB hoạt động với cơ sở dữ liệu Postgres và MySQL chưa được sửa đổi.[[2]](#footnote-2) Việc chuyển CryptDB sang một cơ sở dữ liệu mới rất đơn giản: cổng của chúng tôi sang MySQL chỉ yêu cầu thay đổi 86 dòng mã. Kết quả thử nghiệm của chúng tôi chứng minh rằng CryptDB là di động và áp dụng hình phạt 27% về thông lượng cho khối lượng công việc TPC-C so với DBMS không được mã hóa. Chúng tôi xem chi phí này là tương đối khiêm tốn và là một hình phạt có thể chấp nhận được trong nhiều bối cảnh mà mong muốn bảo mật dữ liệu quan trọng hơn việc đạt được mức hiệu suất cao nhất.

# MÔ HÌNH MỐI ĐE DỌA

Mục tiêu của CryptDB là đảm bảo *tính bí mật*  của dữ liệu trong cơ sở dữ liệu SQL khi đối mặt với *kẻ thù có quyền truy cập hoàn toàn vào máy chủ cơ sở dữ liệu*. Kẻ thù có thể xâm phạm phần mềm máy chủ, hoặc thậm chí tấn công vật lý máy chủ bằng cách đánh cắp và đọc đĩa. Do đó, CryptDB không đưa ra *giả định nào về việc máy chủ cơ sở dữ liệu giữ bất kỳ dữ liệu nào riêng tư*. Mô hình này bao gồm tất cả các kịch bản thúc đẩy mà chúng tôi đã đề cập cho đến nay, chẳng hạn như thuê ngoài cơ sở dữ liệu SQL lên đám mây, bảo vệ chống lại quản trị viên cơ sở dữ liệu tò mò và bảo vệ chống lại những kẻ tấn công đột nhập vào máy chủ cơ sở dữ liệu. Mặt khác, CryptDB giả định rằng ứng dụng và giao diện người dùng CryptDB (Hình 1) không bị xâm phạm và không tiết lộ khóa của chúng cho kẻ thù. Đối phó với các cuộc tấn công vào các thành phần này, chẳng hạn như lỗ hổng SQL injection hoặc các cuộc tấn công bỏ qua xác thực, nằm ngoài phạm vi của bài báo này.

Với mục đích của bài báo này, chúng tôi giả định rằng một máy chủ độc hại không thay đổi dữ liệu hoặc kết quả truy vấn. Đảm bảo tính toàn vẹn cho các truy vấn SQL đã được nghiên cứu rất nhiều và chúng tôi tham khảo các tài liệu trước đây để biết các kỹ thuật để đạt được tính toàn vẹn, như sau. Vì CryptDB cho phép DBMS xử lý các truy vấn quan hệ trên dữ liệu được mã hóa như trên dữ liệu văn bản thuần túy, hầu hết các phương pháp được đề xuất trước đây có thể dễ dàng được sử dụng với CryptDB. Một máy chủ độc hại có thể ảnh hưởng đến ba khía cạnh của bảo mật dữ liệu: tính toàn vẹn, độ mới và tính đầy đủ. Tính toàn vẹn được giải quyết bằng cách thêm MAC vào mỗi bộ như trong [22, 26, 39]. Độ mới đã được giải quyết bằng cách sử dụng băm Merkle hoặc băm chuỗi [22, 36, 39] và cả độ mới và tính đầy đủ của kết quả truy vấn đều được giải quyết trong [32]. Ngoài ra, [25] và [44] cho phép máy khách xác minh kết quả của các truy vấn tổng hợp, [49] xác thực các phép nối và [42] cung cấp đảm bảo truy vấn cho hầu hết các truy vấn đã đọc.

## Định nghĩa bảo mật

Intuition. Ở cấp độ cao, định nghĩa về quyền riêng tư của CryptDB nói rằng CryptDB chỉ tiết lộ mối quan hệ giữa các bộ dữ liệu cần thiết cho máy chủ để thực hiện một số tính toán nhất định; nói cách khác, CryptDB cung cấp *sự riêng tư tối đa cho các lớp tính toán cần thiết tại máy chủ* để xử lý các truy vấn SQL. Cụ thể hơn, định nghĩa nói hai điều:

* (Nếu người dùng không yêu cầu lọc vị ngữ quan hệ trên một cột, tức là họ chỉ yêu cầu phép chiếu và tính toán), không có gì về nội dung cột bị rò rỉ; Nếu người dùng yêu cầu kiểm tra bình đẳng trên một cột, chúng tôi sẽ tiết lộ mục nào lặp lại trong cột đó; Nếu người dùng yêu cầu kiểm tra bất đẳng thức trên một cột, chúng tôi sẽ tiết lộ thứ tự của các phần tử trong cột. Chúng tôi không bao giờ tiết lộ nội dung dữ liệu thực tế.
* Máy chủ không thể xử lý các truy vấn (nghĩa là khám phá các mối quan hệ dữ liệu mới) khác với các truy vấn mà người dùng yêu cầu.

Trực giác này đủ để nắm bắt mô hình bảo mật của CryptDB, vì vậy người đọc không quan tâm đến chủ nghĩa hình thức có thể chuyển sang phần "Ý nghĩa" của phần này bên dưới (và có thể quay lại định nghĩa chính thức về bảo mật sau).

Định nghĩa. Để chính thức hóa các định nghĩa mật mã, thường người ta định nghĩa một thế giới lý tưởng bằng cách sử dụng các nhà tiên tri và sau đó chứng minh rằng giao thức được đề xuất có nhiều quyền riêng tư như *thế giới lý tưởng*  đối với các đối thủ thời gian đa thức. Thay vào đó, chúng tôi sẽ sử dụng thuật ngữ *hệ thống lý tưởng*.

Hệ thống lý tưởng xem xét mục tiêu của chúng tôi là gì? Đây là một hệ thống trong đó máy chủ chỉ học các mối quan hệ dữ liệu mà nó cần để thực hiện xử lý SQL điển hình cho các truy vấn của người dùng và không có gì khác. Điều này tương đương với việc máy chủ không có quyền truy cập vào *nội dung dữ liệu,* mà chỉ với một nhà tiên tri sẵn sàng trả lời các câu hỏi về mối quan hệ giữa các mục dữ liệu, chỉ giới hạn ở các câu hỏi cần thiết để xử lý các truy vấn được yêu cầu.

Cho một truy vấn Q, *hãy để FUNC* (Q*)*  là các lớp tính toán cần thiết cho *Qi*; nghĩa là, mối quan hệ giữa các bộ dữ liệu mà máy chủ được phép biết để xử lý truy vấn. Ví dụ: nếu  Q là SELECT \* FROM table1, table2 WHERE table1.c1 = table2.c2, thì FUNC (*Q*) bao gồm các câu hỏi có dạng: "là mục thứ *i*  trong c1 bằng với mục *thứ j* trong c2". Rõ ràng, máy chủ cần biết thông tin này để xử lý một phép nối, nhưng nó không cần biết giá trị thực tế của hai mục. Trong Phụ lục, chúng tôi mô tả chính xác FUNC.

ĐỊNH NGHĨA 1 (HỆ THỐNG LÝ TƯỞNG). *Hãy để Q1 ... Qt là các truy vấn mà người dùng yêu cầu cho đến thời gian t. Cơ sở dữ liệu tại máy chủ bao gồm từng mục dữ liệu được mã hóa với sơ đồ mã hóa mạnh nhất (ngẫu nhiên). Hơn nữa, máy chủ có quyền truy cập vào một oracle chỉ trả lời các câu hỏi từ* FUNC (*Q1*) ∪ *...* ∪ FUNC (*Qt*)*.*

Rõ ràng, trong Hệ thống lý tưởng, máy chủ có thể xử lý các truy vấn SQL trên dữ liệu được mã hóa vì tất cả thông tin nó cần là các mối quan hệ trong FUNC (*Q1*) ∪ *...* ∪ FUNC (*Qt*), theo định nghĩa của FUNC. Đồng thời, máy chủ như vậy luôn có cơ sở dữ liệu được mã hóa với sơ đồ mã hóa mạnh nhất không rò rỉ gì. Chúng tôi muốn máy chủ trong CryptDB tìm hiểu nhiều thông tin như máy chủ trong Hệ thống lý tưởng và chúng tôi hiển thị thuộc tính này trong §4.

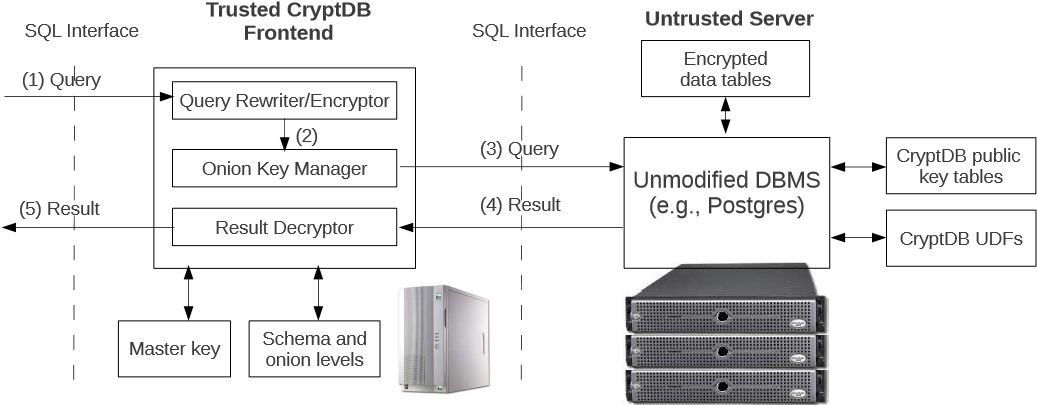
Ý nghĩa. Mặc dù CryptDB không tiết lộ bất kỳ mục dữ liệu nào, nhưng nó không ẩn các mẫu hình truy cập dữ liệu. Ví dụ: máy chủ cơ sở dữ liệu có thể theo dõi tần suất mà một số mục dữ liệu được trả về trong một tập kết quả. Việc che giấu các mẫu hình truy cập như vậy từ máy chủ sẽ phải chịu chi phí đáng kể [33], yêu cầu hầu như ghi đè lên DBMS cơ bản, thêm xử lý phía máy khách đáng kể và ngăn chặn tối ưu hóa phía máy chủ.

Máy chủ cơ sở dữ liệu của CryptDB cũng có thể kiểm tra bất kỳ vị ngữ nào được sử dụng để thực thi các truy vấn SQL do người dùng yêu cầu trong quá khứ, cả tại thời điểm truy vấn được thực thi và tại bất kỳ thời điểm nào sau đó. Ví dụ: nếu ứng dụng-

Khi đưa ra truy vấn SELECT \* FROM table1 WHERE c1 = x2ad412, server sẽ có thể kiểm tra các hàng cho vị ngữ này; Tuy nhiên, điều này sẽ hầu như vô dụng đối với máy chủ vì x2AD412 là một giá trị được mã hóa và tất cả những gì máy chủ học được là số hàng khớp với hằng số chưa biết này.

Lưu ý rằng, khi máy chủ cần đánh giá vị ngữ *P*  (giả sử, id = x245ab1?), máy chủ được phép thực hiện lớp tính toán kiểm tra đẳng thức trên id cột (mặc dù kiểm tra đẳng thức với giá trị được mã hóa) thay vì chỉ tính toán kiểm tra một giá trị cụ thể. Như vậy, máy chủ có thể kiểm tra vị ngữ của dạng c1 = \*, trong đó ∗ là bất kỳ bản mã nào, nhưng có giá trị tương ứng mà máy chủ không biết. Do đó, những gì máy chủ học được là tần suất lặp lại các giá trị chỉ trong cột c1.

Chúng tôi đã xem xét việc cho phép máy chủ chỉ kiểm tra đẳng thức đối với hằng số cụ thể do người dùng đưa ra, nhưng các nguyên thủy mật mã hỗ trợ kiểm tra đẳng thức theo cách này rất tốn kém. Hơn nữa



Hình 1: Tổng quan hệ thống. Các bước (1)-(5) minh họa luồng truy vấn điển hình, với người dùng hoặc ứng dụng đưa ra truy vấn (1) và nhận kết quả (5). UDF là viết tắt của hàm do người dùng xác định.

Làm như vậy sẽ không mang lại nhiều sự riêng tư hơn. Lý do là, thông thường, các truy vấn của người dùng kết thúc thực hiện lựa chọn trên cùng một cột cho nhiều hằng số khác nhau; Sau một vài truy vấn như vậy, quyền riêng tư của chương trình dường như mạnh mẽ hơn này sẽ hội tụ vào quyền riêng tư của chương trình của chúng tôi.

## Bảo mật do người dùng thực thi

Trong một số ứng dụng nhất định, lập trình viên có thể khó tìm thấy tất cả các truy vấn do ứng dụng đưa ra, gây khó khăn cho việc lý luận về các đảm bảo quyền riêng tư chính xác mà CryptDB sẽ cung cấp. Trong trường hợp này, CryptDB cho phép lập trình viên tùy chọn chú thích lược đồ (trong giao diện người dùng, như được định nghĩa trong phần tiếp theo) bằng cách chỉ định mức bảo mật thấp nhất được phép cho mỗi cột. Ví dụ: lập trình viên có thể cho phép kiểm tra đẳng thức, nhưng có thể không muốn tiết lộ trật tự trong cột. Trong trường hợp này, CryptDB sẽ không cho phép máy chủ thực hiện kiểm tra bất bình đẳng và từ chối bất kỳ truy vấn nào yêu cầu tính toán như vậy.

# THIẾT KẾ CRYPTDB

CryptDB hoạt động bằng cách cho phép máy chủ DBMS thực hiện các truy vấn SQL trên dữ liệu được mã hóa gần như thể nó đang thực hiện các truy vấn tương tự trên dữ liệu văn bản thuần túy. Trong CryptDB, kế hoạch truy vấn cho một truy vấn được mã hóa vẫn giống như đối với truy vấn ban đầu, nhưng các hoạt động riêng lẻ bao gồm truy vấn, chẳng hạn như so sánh đẳng thức hoặc tổng hợp, được thực hiện trên bản mã và sử dụng toán tử sửa đổi trong một số trường hợp.

Hình 1 cho thấy kiến trúc của CryptDB. CryptDB bao gồm hai phần: *frontend phía máy khách đáng tin cậy và* máy chủ *DBMS không đáng tin cậy*. Giao diện người dùng theo dõi MK khóa chính bí mật, lược đồ cơ sở dữ liệu như được thấy bởi ứng dụng (tức là không có mã hóa) và mức độ mã hóa hành tây hiện đang được hiển thị tại máy chủ cho từng mục dữ liệu. Mặt khác, máy chủ theo dõi lược đồ được mã hóa, các phiên bản được mã hóa của dữ liệu người dùng (tức là mức mã hóa thấp nhất được tiết lộ cho máy chủ bởi giao diện người dùng) và các bảng phụ trợ được sử dụng bởi CryptDB. Máy chủ triển khai các UDF dành riêng cho CryptDB cho phép frontend tính toán trên các bản mã trên máy chủ. Cuối cùng, máy chủ lưu trữ các phiên bản được mã hóa của trạng thái của giao diện người dùng, bao gồm lược đồ và mức hành tây hiện tại, trong một bảng riêng biệt, được mã hóa bằng MK, cho phép giao diện người dùng trên nhiều máy khách đồng bộ hóa với nhau.

Hình 1 cũng minh họa luồng điển hình của một truy vấn trong CryptDB. Trong bước (1), người dùng đưa ra một truy vấn, trước tiên được chuyển qua trình viết lại / mã hóa truy vấn (QRE). Mô-đun này ẩn danh từng tên bảng và cột. Sử dụng khóa chính MK, mô-đun này mã hóa từng hằng số trong truy vấn bằng sơ đồ mã hóa cho phép hoạt động mong muốn, như chúng tôi sẽ mô tả ngay sau đây. Trong bước (2), truy vấn được chuyển đến trình quản lý khóa củ hành (OKM). Mô-đun này đánh giá xem máy chủ có nên được cung cấp khóa củ hành để thực thi truy vấn hay không. Nếu vậy, OKM cung cấp các khóa hành tây cần thiết bằng cách phát hành truy vấn CẬP NHẬT tại máy chủ gọi UDF để điều chỉnh bảo mật của các cột thích hợp để tăng chức năng của chúng. Trong bước (3), OKM chuyển tiếp truy vấn ẩn danh đến máy chủ, thực thi nó bằng SQL tiêu chuẩn (và đôi khi gọi nhiều UDF hơn, như chúng tôi sẽ giải thích). Trong bước (4), DBMS trả về kết quả truy vấn và trong bước (5), trình giải mã kết quả (RD) giải mã kết quả và trả về cho người dùng. Chúng tôi trình bày chi tiết về các bước này trong phần còn lại của phần này.

## Chiến lược mã hóa nhận biết SQL

Để thực hiện mã hóa cho phép xử lý truy vấn SQL, chúng tôi sử dụng các sơ đồ mã hóa hiện có, tối ưu hóa sơ đồ gần đây và thiết kế một nguyên thủy mật mã mới cho các phép nối, như chúng tôi sẽ mô tả bây giờ. CryptDB sử dụng cùng một mức mã hóa để mã hóa tất cả các mục dữ liệu trong một cột nhất định, do đó cùng một tính toán có thể được thực hiện trên mọi phần tử trong cột đó.

Đối với mỗi loại mã hóa, chúng tôi giải thích thuộc tính bảo mật mà CryptDB yêu cầu từ nó. Chúng tôi giải thích cách triển khai nó với các công cụ được cho là đạt được bảo mật như vậy; nếu các công cụ như vậy bị hỏng, chúng có thể được thay thế bằng các công cụ khác cung cấp tài sản đó mà không phá vỡ thiết kế bảo mật của CryptDB.

Ngẫu nhiên (RND). RND cung cấp sự riêng tư tối đa, chẳng hạn như không thể phân biệt được theo cuộc tấn công bản mã được chọn thích ứng (INDCCA2), còn được gọi là bảo mật ngữ nghĩa. Đặc biệt, hai giá trị bằng nhau sẽ được ánh xạ tới các mã hóa khác nhau với xác suất cao. RND không cho phép bất kỳ tính toán nào được thực hiện hiệu quả trên bản mã. Để triển khai RND, chúng tôi sử dụng AES ở chế độ UFE [11].

Xác định (DET). DET có đảm bảo yếu hơn một chút: nó chỉ rò rỉ các giá trị được mã hóa tương ứng với cùng một giá trị dữ liệu và không có gì khác. Mức mã hóa này cho phép máy chủ thực hiện kiểm tra bình đẳng, có nghĩa là nó có thể thực hiện các lựa chọn với các bộ lọc đẳng thức, tham gia bình đẳng, GROUP BY, COUNT, DISTINCT, v.v. Có nhiều cách để thực hiện DET, chẳng hạn như DETK (v) = *RNDK1* (v) k HMAC − SHA1K2 (v)*,* trong đó k là toán tử ghép nối, K1 và *K2* là hai khóa có nguồn gốc từ  *K* và *K*  bản thân nó có nguồn gốc bằng cách mã hóa tên bảng và cột bằng MK. Đối với cấu trúc DET này, máy chủ so sánh hai mã hóa bằng

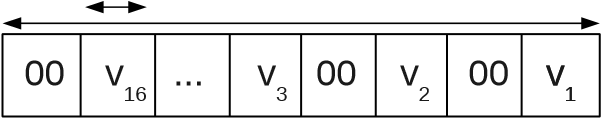
so sánh các giá trị HMAC−SHA1 của chúng.(OPE). OPE cho phép mã hóa bảo quản đơn hàng giữa các mục dữ liệu được thiết lập dựa trên giá trị được mã hóa của chúng, nhưng không rò rỉ bất kỳ thông tin nào khác về nội dung dữ liệu. Một đề xuất gần đây cho OPE [3] là một sơ đồ mã hóa duy trì trật tự: nếu x < y*, thì OPEK (*x) *<* OPEK *(*y), *cho bất kỳ khóa bí mật nào* K. Do đó, nếu một cột được mã hóa bằng OPE, máy chủ có thể thực hiện các truy vấn phạm vi khi được cung cấp các hằng số được mã hóa OPEK (c1) *và OPEK* (c2 ) *biểu thị phạm vi* từ c1 đến c2. Hơn nữa, máy chủ có thể thực hiện ORDER BY, MIN, MAX, SORT, v.v.

OPE là một sơ đồ mã hóa yếu hơn vì nó tiết lộ trật tự. Do đó, giao diện người dùng sẽ chỉ hiển thị các cột được mã hóa OPE cho máy chủ nếu người dùng yêu cầu truy vấn phạm vi trên các cột đó. OPE có đảm bảo bảo mật có thể chứng minh: mã hóa tương đương với hoán vị ngẫu nhiên để duy trì trật tự. Do đó, sự khác biệt giữa hai mã hóa OPEK (y) - *OPEK* (x) *về cơ bản là ngẫu nhiên và không bằng* y -lược đồ chỉ được đề xuất vào năm ngoái [3]. *x* ngoại trừ dấu hiệu.

Một OPE an toàn có thể chứng minh được

Không có việc thực hiện đề án, hoặc bất kỳ biện pháp nào về mức độ thực tế của nó, vì vậy chúng tôi đã thực hiện nó. Hiệu suất ban đầu là khoảng 20 ms cho mỗi mã hóa, điều này không tuyệt vời, vì mỗi hàng trong bảng có thể có một vài mục cần được mã hóa bằng OPE. Chúng tôi đã thực hiện một số tối ưu hóa để giảm chi phí OPE. Ở cấp độ cao, cho giá trị *v*, OPE thực hiện

ABC D ABC



Hình 2: Đóng gói 16 giá trị số nguyên (mỗi giá trị 32 bit) thành một giá trị 1024 bit duy nhất sau đó được mã hóa bằng HOM.

Tìm kiếm nhị phân trong lĩnh vực mã hóa để tìm mã hóa cho *v*. Chúng tôi nhận ra rằng kết quả tìm kiếm trung gian có thể được sử dụng lại trên các mã hóa có giá trị khác nhau, vì vậy chúng tôi lưu chúng vào bộ nhớ cache khi thực hiện nhiều mã hóa như vậy (ví dụ: khi tải cơ sở dữ liệu). Để tìm kiếm bộ nhớ cache một cách hiệu quả, chúng tôi đã sử dụng cây tìm kiếm nhị phân AVL nhanh. Điều này làm giảm chi phí mã hóa OPE xuống còn 4 ms, mà không ảnh hưởng đến bảo mật của nó. Chúng tôi cũng đã triển khai một bộ lấy mẫu siêu hình học nằm ở cốt lõi của OPE. Sơ đồ hiệu quả nhất như vậy đã được đề xuất vào năm 1988 (hiện được sử dụng trong nhiều công cụ như MathWorks) [21] và chúng tôi đã dịch triển khai ban đầu của nó từ Fortran-1988.

Mã hóa đồng cấu (HOM). HOM là một sơ đồ mã hóa an toàn IND-CCA, nhưng cho phép máy chủ thực hiện tính toán trên dữ liệu được mã hóa, kết quả cuối cùng được giải mã bởi giao diện người dùng. Trong khi mã hóa đồng cấu cho các hoạt động chung rất chậm [9], mã hóa đồng cấu để tổng hợp là hiệu quả. Để hỗ trợ tổng kết, chúng tôi đã triển khai hệ thống mật mã Paillier [35]. Với Paillier, nhân mã hóa của hai giá trị dẫn đến mã hóa tổng các giá trị, tức là,

HOMis thực hiện modulo một số giá trị khóa công khai. Khi máy chủ per-K(*x*) · HOMK(y) = HOMK(x *+* y*)*, trong đó phép nhân

hình thức tổng kết trên một cột được mã hóa bằng HOM, nó gọi một UDF thực hiện phép nhân Paillier và sử dụng bảng khóa công khai của CryptDB để tra cứu *K*, để tính toán đồng cấu một bản mã tương ứng với tổng các giá trị bản rõ. HOM được bảo mật dưới một cuộc tấn công văn bản rõ ràng đã chọn. HOM cũng có thể được sử dụng để tính toán trung bình (bằng cách yêu cầu máy chủ trả về tổng và đếm, và chia tổng được giải mã cho số đếm trong giao diện người dùng) và để tăng các giá trị (ví dụ: SET id = id + 1), mà chúng tôi sẽ trình bày chi tiết ngay sau đây.

Một nhược điểm với HOM là độ dài bản mã dài 2048 bit cho mỗi giá trị dữ liệu. Tuy nhiên, chúng tôi đã thực hiện quan sát sau: một hàng chỉ có thể lưu trữ một bản mã HOM cho một số cột số nguyên, bởi vì mỗi bản mã HOM 2048 bit tương ứng với giá trị bản rõ 1024 bit và 8 giá trị 32 bit từ các cột khác nhau có thể được đóng gói thành một giá trị 1024 bit ở các độ lệch bit khác nhau (cho phép 32 không bit giữa mỗi giá trị số nguyên 32 bit). Các cột riêng lẻ vẫn có thể được tổng hợp bằng cách yêu cầu giao diện người dùng trích xuất tổng các cột mong muốn từ phần bù thích hợp trong bản rõ được tính toán đồng cấu. Hình 2 minh họa sự tối ưu hóa này.

Tìm kiếm từ (TÌM KIẾM). Để cho phép tìm kiếm từ (ví dụ: sử dụng từ khóa "ILIKE"), chúng tôi triển khai giao thức mật mã cho các tìm kiếm từ khóa trên văn bản được mã hóa [1, 43]. SEARCH cho phép máy chủ phát hiện các từ lặp lại trong một cột nhất định.

Tham gia (JOIN và OPE−JOIN). Một mức mã hóa riêng biệt là cần thiết để cho phép tham gia bình đẳng giữa hai cột, bởi vì mức mã hóa DET sử dụng các khóa khác nhau cho mỗi cột. Ở cấp độ cao, JOIN cho phép máy chủ so sánh các giá trị giữa các giá trị trong cột *A* và *B*, được cung cấp một mã thông báo từ giao diện người dùng cho các cột

*A* và *B*. JOIN cũng hỗ trợ tất cả các hoạt động được cho phép bởi DET và SEARCH, và cũng cho phép máy chủ phát hiện các giá trị lặp lại giữa hai cột. Đối với các phép tham gia bất đẳng thức, OPE−JOIN cho phép máy chủ thực hiện các phép nối bất đẳng thức trên bất kỳ cột nào ở cấp mã hóa đó. Cơ chế của các cấp độ mã hóa này được thảo luận chi tiết hơn trong §3.5, cùng với toàn bộ cơ chế nối.

bất kỳ giá trị nào

THAM GIA: tham gia bình đẳng

TÌM KIẾM: tìm kiếm từ

)

Chỉ dành cho các trường văn bản

(

DET: lựa chọn bình đẳng

bất kỳ giá trị nào

OPE-JOIN: tham gia bất bình đẳng

OPE: so sánh

RND: không có chức năng

Giá trị INT

HOM: tổng

Hành tây 1

Hành tây 2

Hành tây 3

RND: không có chức năng

Hình 3: Các lớp mã hóa củ hành tây và các lớp tính toán mà chúng cho phép.

## Mã hóa dựa trên truy vấn có thể điều chỉnh

Một phần quan trọng trong thiết kế của CryptDB là  *mã hóa dựa trên truy vấn có thể điều chỉnh*, xác định động mức độ mã hóa để tiết lộ cho máy chủ. Câu trả lời phụ thuộc vào các truy vấn được hỏi qua dữ liệu: nếu không có lý do gì để so sánh các mục dữ liệu trong một cột hoặc sắp xếp một cột, cột sẽ được mã hóa bằng RND và đối với các cột thực hiện kiểm tra bình đẳng nhưng không kiểm tra bất bình đẳng, DET là đủ. Thật không may, có nhiều cơ sở dữ liệu mà bộ truy vấn không được biết trước. Do đó, chúng ta cần một sơ đồ thích ứng tự động "làm điều đúng đắn" về mặt lựa chọn chiến lược mã hóa cho truy vấn trong tầm tay.

Ý tưởng của chúng tôi là mã hóa từng ô một cách độc lập thành một *củ hành*: mỗi giá trị trong bảng được bọc trong các lớp mã hóa ngày càng mạnh hơn, như minh họa trong Hình 3. Mỗi lớp của mỗi củ hành cho phép một số loại chức năng nhất định như được giải thích trong tiểu mục trước. Ví dụ, các lớp ngoài cùng, RND và HOM, cung cấp sự riêng tư tối đa, trong khi OPE cung cấp nhiều chức năng hơn. Đối với các giá trị số, CryptDB duy trì ba củ hành, trong khi đối với các giá trị chuỗi, CryptDB duy trì hai củ hành (tức là không có HOM). Để ngăn máy chủ học thông tin từ tên cột hoặc bảng, giao diện người dùng của CryptDB cũng ẩn danh lược đồ. Hình 4 cho thấy một lược đồ và dữ liệu phía máy chủ ví dụ trong CryptDB, cùng với các bản rõ tương ứng. Mỗi mục dữ liệu được lưu trữ mã hóa trong tối đa 3 củ hành (đối với số nguyên) hoặc 2 củ hành (đối với các giá trị không phải số nguyên), mặc dù các tối ưu hóa của CryptDB có thể giảm xuống còn 1 củ hành cho một số bộ truy vấn (như đã thảo luận trong §3.7).

Đối với mỗi cấp độ của mỗi củ hành, giao diện người dùng sử dụng cùng một khóa để mã hóa các giá trị trong cùng một cột và các khóa khác nhau trên các cột, cấp hành tây và bảng. Tất cả các khóa này có nguồn gốc từ khóa chính MK. Ví dụ: đối với bảng *t*, cột *c*, mức mã hóa *l*, giao diện người dùng sử dụng khóa

*Kt, c, l* = PRPMK("Bàn *t*", "cột *c*", "cấp độ *l*")*,* (1)

trong đó PRP là hoán vị giả ngẫu nhiên (chẳng hạn như AES).

Mỗi củ hành bắt đầu được mã hóa bằng sơ đồ mã hóa riêng tư nhất (RND cho hành tây 1 và 2 và HOM cho hành tây 3). Khi frontend nhận các truy vấn SQL từ ứng dụng, trình quản lý khóa hành tây (OKM) xác định xem có cần xóa các lớp mã hóa hay không. Cho một vị ngữ P trên cột c cần thiết *cho truy vấn, OKM trước tiên thiết lập lớp hành tây nào là cần thiết để thực hiện* P *trên*  c. Nếu mã hóa *c* đã ở một lớp hành tây cho phép *P*, OKM không làm gì cả. Nếu không, OKM phải loại bỏ các lớp hành tây để cho phép *P* trên *c*, bằng cách gửi khóa củ hành tương ứng đến máy chủ.

Để tránh thay đổi DBMS, CryptDB thực hiện giải mã lớp hành tây bằng cách sử dụng các chức năng do người dùng xác định trong máy chủ. Ví dụ: để giải mã hành tây 2 của cột 2 trong bảng 1 để cấp OPE, OKM đưa ra truy vấn sau cho máy chủ, sử dụng DECRYPT\_RND UDF:

### ABC D

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | | EF | C | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | ABCDB | ABCDBE | ABCDBF | E ABCDB | E ABCDBE | | E E |  | EF | F | F | |

Hình 4: Bố trí dữ liệu tại máy chủ. Khi frontend tạo một bảng với lược đồ ở bên trái, bảng được tạo tại máy chủ là bảng từ bên phải.

CẬP NHẬT Bảng 1 SET C2-Onion2 =

DECRYPT\_RND(K, C2-Hành tây2)

trong đó *K* là khóa thích hợp được tính như trong Eq. (1).

Lưu ý rằng việc giải mã được thực hiện hoàn toàn bởi máy chủ chứ không phải bởi máy khách. Quan trọng hơn, ở trạng thái ổn định, không cần giải mã phía máy chủ, vì giải mã cột chỉ xảy ra khi một loại vị ngữ mới được thực hiện trên một cột. Ví dụ: sau khi vị ngữ đẳng thức được thực hiện trên một cột và máy chủ đưa cột đến mức DET, cột vẫn ở trạng thái đó và các truy vấn như vậy trong tương lai không còn yêu cầu giải mã nữa. Như §6 sẽ minh họa, điều này đảm bảo rằng, ở trạng thái ổn định, chi phí của CryptDB thấp có thể chấp nhận được. Mức độ riêng tư kết quả mà cơ sở dữ liệu hội tụ là mức bảo mật tối đa cho tập hợp các truy vấn do ứng dụng ban hành.

## Thực hiện truy vấn cho truy vấn đọc

Để thực thi mỗi truy vấn SQL, frontend ẩn danh, mã hóa và viết lại truy vấn trước khi chuyển tiếp nó đến máy chủ DBMS không đáng tin cậy. Để ẩn danh một truy vấn, frontend thay thế mỗi bảng bằng tên bảng từ lược đồ ẩn danh (xem Hình 4). Để chiếu, giao diện người dùng thay thế mỗi cột bằng tên của cột ẩn danh cho củ hành đầu tiên. Để thực hiện vị ngữ hoặc tổng hợp trên một cột, giao diện người dùng thay thế cột bằng tên ẩn danh của củ hành cho phép thao tác cần thiết trên trường đó và đối với một số hoạt động nhất định (chẳng hạn như SUM), giao diện người dùng thay thế hoạt động bằng UDF tương đương hoạt động trên bản mã. Cuối cùng, mỗi hằng số trong truy vấn được mã hóa bằng sơ đồ mã hóa tương ứng với lớp hành tây được sử dụng bởi vị ngữ liên quan đến hằng số.

Để minh họa cách thức hoạt động của điều này, hãy xem xét một kịch bản ví dụ bao gồm bảng Nhân viên, có bốn cột quan tâm: id, tên, địa chỉ và tiền lương. Ban đầu, mỗi cột trong bảng được mặc tất cả các củ hành mã hóa với RND và HOM là các lớp ngoài cùng, như thể hiện trong Hình 3. Tại thời điểm này, máy chủ không thể tìm hiểu gì về nội dung dữ liệu ngoài số cột, hàng và kích thước dữ liệu.

Để minh họa khi nào các lớp hành tây được loại bỏ, hãy xem xét truy vấn

CHỌN \* TỪ Nhân viên WHERE name = 'Alice', yêu cầu giảm mã hóa tên xuống cấp DET. Trong trường hợp này, frontend đầu tiên đưa ra truy vấn UPDATE Table1 SET

C2-Onion1 = DECRYPT\_RND(K1,2,RND, C2-Onion1), và sau đó CHỌN C1-Onion1, C2-Onion1, *C3-Onion1, C4-Onion1 TỪ Bảng 1 TRONG ĐÓ C2-Onion1 =*

x7d35a3, trong đó x7d35a3 là mã hóa của "Alice" bằng khóa

*K1,2,DET*. Giao diện người dùng giải mã kết quả từ máy chủ và trả lại cho người dùng.

Nếu truy vấn tiếp theo là SELECT COUNT(\*) TỪ Nhân viên WHERE name = 'Bob', không cần giải mã phía máy chủ bổ sung và giao diện người dùng trực tiếp đưa ra truy vấn

CHỌN COUNT(\*) TỪ Bảng 1 WHERE C2-Onion1

= xbb234a, trong đó xbb234a là mã hóa của "Bob".

Cuối cùng, giao diện người dùng thay thế các toán tử tổng hợp bằng các UDF tương đương hoạt động trên các giá trị được mã hóa. Ví dụ: nếu người dùng đưa ra truy vấn SELECT SUM(salary) FROM Employees, frontend sẽ viết lại thành SELECT HOM\_AGG(C4-Onion3, PKTABLE. PK) TỪ Table2, trong đó HOM\_AGG là một UDF thực hiện phép nhân Paillier (dẫn đến việc bổ sung các bản rõ), PKTABLE là một bảng các khóa công khai mà CryptDB lưu trữ tại máy chủ (xem Hình 1) và PK là mô đun khóa công khai Paillier.

## Thực thi truy vấn để ghi truy vấn

CryptDB cũng hỗ trợ các truy vấn sửa đổi dữ liệu trên máy chủ— cụ thể là INSERT, DELETE và UPDATE. Đối với tất cả các truy vấn sửa đổi, giao diện người dùng áp dụng cùng một xử lý cho các vị ngữ (tức là mệnh đề WHERE) như đối với các truy vấn đọc. Đối với các truy vấn INSERT, frontend mã hóa giá trị của từng cột được chèn với mỗi lớp hành tây chưa bị tước bỏ trong cột đó. Đối với truy vấn DELETE, không có xử lý bổ sung nào được thực hiện. Đối với các truy vấn UPDATE đặt giá trị của cột thành hằng số, giao diện người dùng sẽ mã hóa giá trị trong củ hành thích hợp như đối với INSERT.

Trường hợp còn lại là truy vấn CẬP NHẬT cập nhật giá trị cột dựa trên giá trị cột khác, chẳng hạn như lương = lương + 1. Một bản cập nhật như vậy sẽ phải được thực hiện bằng HOM, bởi vì nó cho phép bổ sung. Tuy nhiên, khi làm như vậy, các giá trị trong hành OPE và DET sẽ trở nên cũ kỹ. Trên thực tế, một sơ đồ mã hóa cho phép cả cộng và so sánh cùng một lúc về cơ bản là không an toàn: nếu một máy chủ độc hại biết thứ tự của các mục (OPE) và có thể tăng giá trị lên một, máy chủ có thể tiếp tục thêm một vào mỗi trường đồng cấu cho đến khi trường trở nên bằng một giá trị khác. Điều này sẽ cho phép máy chủ tính toán sự khác biệt giữa hai giá trị bất kỳ trong cơ sở dữ liệu, gần như tương đương với việc biết giá trị của chúng.

Có hai giải pháp cho vấn đề này. Nếu một cột được tăng lên và sau đó chỉ được chiếu (không có so sánh nào được thực hiện trên đó), giải pháp rất đơn giản: khi yêu cầu giá trị của trường này, hãy sử dụng giá trị của Hành tây 3 thay vì Hành tây 1 hoặc 2, vì Hành tây 3 được cập nhật. Đây là trường hợp cho hầu hết các truy vấn TPC-C. Nếu một cột được sử dụng trong so sánh sau khi nó được tăng, giải pháp là chia truy vấn thành hai. Đó là, đối với truy vấn CẬP NHẬT Nhân viên SET lương = lương + 1 TRONG ĐÓ id = 3, giao diện người dùng phát hành phiên bản được mã hóa của CHỌN lương TỪ Nhân viên WHERE id = 3, và sau đó là phiên bản được mã hóa của UPDATE

Nhân viên SET lương = z WHERE id = 3, trong đó z nhiều hơn một so với kết quả của truy vấn đầu tiên.[[3]](#footnote-3) Tuy nhiên, trong hầu hết các trường hợp trong thực tế (chẳng hạn như trong TPC-C), các cập nhật như vậy được thực hiện trên các hàng riêng lẻ.

## Máy tính tham gia

Hỗ trợ tham gia là một vấn đề đầy thách thức. Nếu hai cột được nối, chúng cần được mã hóa bằng cùng một khóa cho các cấp

JOINdeterministic joins (trường hợp phổ biến áp đảo, tức là levelor OPE−JOIN. Trước tiên, chúng tôi mô tả cách CryptDB triển khai

JOINTo cung cấp sự riêng tư tối đa cho các tham gia bình đẳng, máy chủ nên), và sau đó mô tả các tham gia bất bình đẳng (mức OPE−JOIN). không thể nối các cột mà người dùng không yêu cầu nối, vì vậy các cột không bao giờ được nối không nên được mã hóa bằng cùng một khóa JOIN mật mã. Hơn nữa, nếu người dùng yêu cầu nối các cột *A*  và B và nối các cột  *C và D*, máy chủ sẽ không thể nối *B* và *C* . Do đó, câu hỏi đặt ra là, mỗi cột nên được mã hóa bằng khóa JOIN nào, vì chúng ta không biết trước những cột nào sẽ được nối?

Để giải quyết vấn đề này, chúng tôi đề xuất một nguyên thủy mật mã mới cho phép máy chủ tự động điều chỉnh các khóa mã hóa JOIN của mỗi cột. Mỗi cột ban đầu được mã hóa bằng một khóa JOIN khác nhau, do đó không cho phép tất cả các phép nối. Khi người dùng yêu cầu nối, giao diện người dùng sẽ cung cấp cho máy chủ một khóa củ hành để mã hóa lại hai cột thành cùng một khóa JOIN, cho phép nối giữa hai cột.

Thuật toán của chúng tôi dựa trên mật mã đường cong elip (ECC). Khi một hàng ban đầu được chèn, mã hóa JOIN của giá trị v *được tính là JOINK* (v) : =  *H (*v*)* K, trong  *đó* K *là* khóa ban đầu cho bảng, cột và mức đó và *H* là ánh xạ từ các giá trị (số nguyên hoặc chuỗi) đến đường cong elip. Khi người dùng yêu cầu nối các cột c và c0, *giao diện người dùng sẽ tính toán* ∆K = *K / K0*, có thể được sử dụng để đưa mã hóa JOIN của c và *c0* vào cùng một khóa. Cho JOINK0(v) (được lưu trữ trong cột *c0*) và ∆K, máy chủ sử dụng UDF để tính JOINK0(v)∆K = H(v)K0×K/K0  *= H*(*v)*K  *=* JOINK*(v)*. Bây giờ các cột *c* và *c0* chia sẻ như nhau

Khóa JOIN, máy chủ có thể thực hiện nối đẳng thức trên *c* và *c0* như bình thường.

ĐỊNH LÝ 1. *Máy chủ chỉ có thể nối các cặp cột được nối với truy vấn người dùng hợp pháp.*

PHÁC THẢO BẰNG CHỨNG. Chúng tôi ủy thác bằng chứng đầy đủ cho phiên bản mở rộng của bài báo này và chỉ cung cấp một bản phác thảo ở đây. Lược đồ của chúng tôi là an toàn (nghĩa là định lý là đúng) vì  *K / K0 không tiết lộ bất kỳ thông tin nào về* K *hoặc* K0 một mình. Hơn nữa, máy chủ không thể tính K từ *H* (*v*) *K* vì giả định độ cứng của việc tính toán logarit rời rạc trong các nhóm dựa trên đường cong elip. Với hai cột được mã hóa bằng K *và K0,* nếu máy chủ không được cung cấp *K / K0*, nó không thể đưa hai cột đến cùng một mức mã hóa và do đó không thể thực hiện các phép nối không được yêu cầu.

Chúng ta Chọn ECC vì của của nó hiệu quả: điện toán

JOINK0 (*v*) *K / K0* về cơ bản là phép nhân và không liên quan đến bất kỳ số mũ nào. Hơn nữa, các bản mã dựa trên đường cong elip là nhỏ (các hệ thống mật mã cho cùng một mức độ bảo mật [31].≈ 160 bit) so với bản mã trong khóa công khai điển hình

Đối với các tham gia bất bình đẳng, một sơ đồ mã hóa lại động tương tự rất khó xây dựng. Thay vào đó, CryptDB yêu cầu các cặp cột sẽ tham gia vào các tham gia bất đẳng thức được các ứng dụng khai báo trước thời hạn, bằng cách chú thích lược đồ, để các khóa phù hợp được sử dụng cho mức OPE−JOIN của các cột đó. Ngoài ra, mã hóa OPE−JOIN cấp có thể được mã hóa lại bằng các khóa phù hợp với chi phí gửi toàn bộ cột đến giao diện người dùng để mã hóa lại, sau đó gửi lại máy chủ DBMS.

## Giao dịch và Chỉ số

CryptDB sử dụng các cơ chế giao dịch và lập chỉ mục hiện có trong máy chủ DBMS mà không có bất kỳ sửa đổi nào. Đối với các giao dịch, giao diện người dùng chuyển mọi truy vấn BEGIN, COMMIT, và ABORT đến DBMS và CryptDB không thay đổi bất kỳ ngữ nghĩa giao dịch nào. Đối với các chỉ mục, DBMS xây dựng các chỉ mục của các cột được mã hóa giống như cách nó xây dựng các chỉ mục của dữ liệu văn bản thuần túy. Giao diện người dùng không yêu cầu chỉ mục trên mã hóa RND, vì không có tra cứu nào được thực hiện ở cấp độ đó, nhưng giao diện người dùng xây dựng các chỉ mục trên mã hóa DET, JOIN, OPE và OPE−JOIN, nếu  ứng dụng yêu cầu chỉ mục trên cột tương ứng trong lược đồ gốc (ví dụ: sử dụng truy vấn ALTER hoặc CREATE).

## Tối ưu hóa

CryptDB thực hiện ba tối ưu hóa quan trọng để cải thiện hiệu suất của nó, mà bây giờ chúng tôi mô tả.

Tập hợp truy vấn đã biết. Đối với nhiều ứng dụng, các truy vấn do ứng dụng đưa ra là cố định và được biết trước. Trong trường hợp này, chúng ta không cần điều chỉnh mức hành tây trong thời gian chạy và có thể khởi động cơ sở dữ liệu với các mức mã hóa củ hành chính xác mà chúng ta cần. Hơn nữa, nếu một cột nhất định không bao giờ được sử dụng trong một số hoạt động nhất định, hành tây tương ứng có thể được bỏ qua hoàn toàn. Ví dụ: nếu một ứng dụng không bao giờ thực hiện các thao tác phạm vi hoặc thứ tự trên một cột, hành tây OPE cho cột đó có thể được bỏ qua. Loại bỏ hành tây làm giảm đáng kể chi phí hiệu suất cả trong giao diện người dùng và trong máy chủ DBMS. Tối ưu hóa này được thực hiện bởi  *một* mô-đun tàu ở giao diện người dùng, với một tập hợp các truy vấn mà ứng dụng người dùng sẽ đưa ra, xác định mức độ mã hóa chính xác cần thiết.

Hội tụ an ninh. Ngay cả khi bộ truy vấn không được biết trước, sau khi một ứng dụng đã chạy trong một thời gian đáng kể trên DB, giao diện người dùng có thể bỏ bất kỳ củ hành nào chưa được sử dụng, vì những củ hành này khó có thể được sử dụng trong tương lai. Nếu một truy vấn xảy ra để sử dụng các củ hành này sau đó, giao diện người dùng có thể đi qua chi phí chèn cột đã xóa, bằng cách tải xuống toàn bộ cột và tải lên lại trong một củ hành khác, được mã hóa lại. Vì các hoạt động này được khấu hao trong thời gian dài, hiệu suất tổng thể vẫn cao.

Bộ nhớ đệm bản mã. Một chi phí liên tục đáng kể cho giao diện người dùng nằm trong việc tạo mã hóa OPE và HOM của các hằng số được sử dụng trong các truy vấn. Để tránh chi phí này, giao diện người dùng duy trì bộ nhớ cache của các hằng số được sử dụng gần đây, cùng với mã hóa của chúng dưới các khóa khác nhau. Vì một số hằng số được sử dụng nhiều lần trong nhiều truy vấn (ví dụ: hằng số 1), việc tối ưu hóa này làm giảm lượng thời gian CPU dành cho dữ liệu mã hóa giao diện người dùng.

## Thảo luận và hạn chế

Thiết kế của CryptDB hỗ trợ hầu hết các truy vấn quan hệ và tổng hợp trên các kiểu dữ liệu tiêu chuẩn, chẳng hạn như số nguyên và kiểu văn bản / varchar. Dữ liệu số thuộc loại thập  *phân có chữ số p* sau số thập phân có thể được ánh xạ tới các loại số nguyên bằng cách nhân đầu vào với 10p. CryptDB có thể mã hóa các giá trị dấu phẩy động, nhưng không thể thực hiện tổng hợp trên các giá trị dấu phẩy động (tuy nhiên, nếu các giá trị được chuyển đổi thành điểm cố định, CryptDB có thể sử dụng HOM để tổng hợp). Các hoạt động bổ sung có thể được thêm vào CryptDB bằng cách mở rộng các củ hành hiện có của nó hoặc thêm củ hành mới cho các loại dữ liệu cụ thể. Ví dụ: người ta có thể thêm các truy vấn phạm vi không gian và đa chiều bằng cách sử dụng các giao thức được đề xuất bởi Shi et al [40].

CryptDB có những hạn chế nhất định. Ví dụ: nó không hỗ trợ cả tính toán và so sánh *trong cùng một vị ngữ*, chẳng hạn như WHERE lương > tuổi \* 2 + 10. CryptDB có thể tạo điều kiện thuận lợi cho việc xử lý một truy vấn như vậy, nhưng nó sẽ yêu cầu một chút xử lý trên giao diện người dùng. Để sử dụng CryptDB, truy vấn có thể được viết lại thành một truy vấn con chọn toàn bộ cột, SELECT age\*2+10 FROM *...*, mà CryptDB tính toán bằng HOM, sau đó mã hóa lại kết quả trong giao diện người dùng, tạo một cột mới (gọi nó là aux) tại máy chủ bao gồm các giá trị mới được mã hóa và cuối cùng chạy truy vấn ban đầu với vị ngữ WHERE lương > aux.

Nguyên mẫu CryptDB hiện tại không hỗ trợ các thủ tục được lưu trữ hoặc các chức năng do người dùng xác định khác trên máy chủ. Hỗ trợ các thủ tục được lưu trữ được viết bằng SQL nên đơn giản, bằng cách yêu cầu giao diện người dùng viết lại các câu lệnh SQL bên trong quy trình được lưu trữ như đối với bất kỳ truy vấn nào khác. Mặt khác, thiết kế của CryptDB không thể hỗ trợ thực thi các hàm do người dùng xác định tùy ý (không được viết bằng SQL) trên dữ liệu được mã hóa.

Cuối cùng, nguyên mẫu hiện tại của chúng tôi xử lý các cột tự động tăng phía máy chủ bằng cách để các giá trị cột ở dạng bản rõ trên máy chủ. Chúng tôi tin rằng đây là một sự đánh đổi có thể chấp nhận được, vì máy chủ dù sao cũng tham gia vào việc chọn giá trị tự động tăng. Một sơ đồ bảo vệ quyền riêng tư hơn có thể liên quan đến việc sử dụng HOM để tạo giá trị tự động tăng, nhưng HOM sẽ không cho phép nối trên cột đó (trong khi tự động tăng thường được sử dụng cho các cột khóa chính yêu cầu nối). Tổng quát hơn, CryptDB cho phép các cột không nhạy cảm với quyền riêng tư không được mã hóa trên máy chủ, để giảm chi phí và cho phép tính toán tổng quát hơn (chẳng hạn như UDF tùy ý).

# PHÂN TÍCH QUYỀN RIÊNG TƯ

Trong phần này, chúng tôi chứng minh rằng CryptDB đạt được mục tiêu bảo mật mà chúng tôi đặt ra trong §2. Để làm như vậy, chúng tôi giả định rằng các công cụ mật mã được sử dụng bởi CryptDB thực sự cung cấp các thuộc tính mà chúng tôi yêu cầu. Các cấu trúc mà chúng tôi đề xuất khi triển khai trong phần trước là an toàn dựa trên một số giả định mật mã; nếu các giả định như vậy bị phá vỡ, các cấu trúc mật mã có thể được thay thế bằng các cấu trúc khác có cùng đảm bảo bảo mật, do đó bảo vệ tính bảo mật của thiết kế CryptDB tổng thể.

ĐỊNH LÝ 2. *Máy chủ trong CryptDB tìm hiểu nhiều thông tin về dữ liệu trong cơ sở dữ liệu cũng như máy chủ trong Hệ thống lý tưởng từ Định nghĩa 1.*

PHÁC THẢO BẰNG CHỨNG. Bằng chứng là bằng cách quy nạp các truy vấn được yêu cầu. Trường hợp cơ bản của quy nạp là khi không có truy vấn nào được yêu cầu cho đến nay. Vì chúng tôi bắt đầu cơ sở dữ liệu với RND, chúng tôi đáp ứng trường hợp cơ sở.

Trong bước quy nạp, chúng ta cần chứng minh rằng, bằng cách xử lý *truy vấn* thứ i*, Qi, máy chủ chỉ học được nhiều thông tin như các mối quan hệ có trong FUNC* (Qi*).* Bằng chứng như vậy là bằng cách xem xét toàn diện tất cả các loại hoạt động; Để ngắn gọn, chúng tôi chỉ xem xét những cái cơ bản nhất ở đây, để phần còn lại cho phiên bản mở rộng của bài báo này. Giả sử *Qi* chứa vị ngữ đẳng thức: c1 = x1c5a21. Sau đó, FUNC (*Qi*) chứa mối quan hệ {c1 = \*}. CryptDB sẽ hạ thấp mức c1 xuống DET. Theo giả định về mức độ bảo mật do DET cung cấp, máy chủ chỉ có thể kiểm tra xem một số dữ liệu được mã hóa từ c1 có bằng một số dữ liệu được mã hóa khác từ c1 hay không, tất cả đều đã được c1 = \* từ FUNCc1 (*Qi*) cho phép vì máy chủ có thể thay thế ∗ bằng bất kỳ giá trị nào từ

.

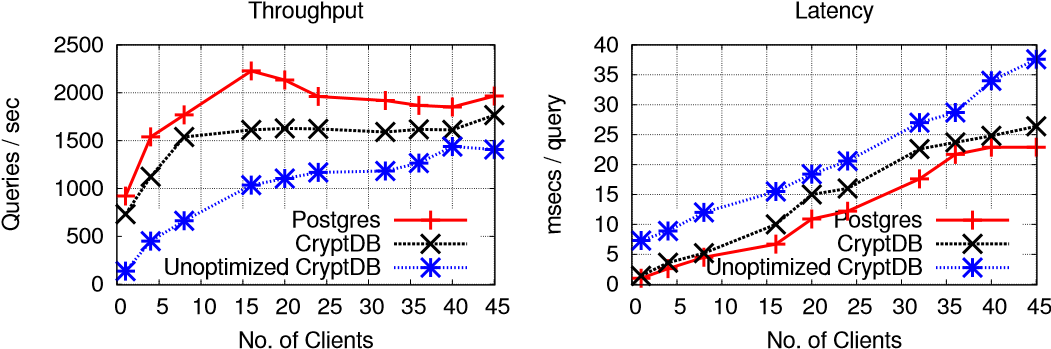
Bây giờ hãy xem xét rằng *Qi*  chứa kiểm tra bất đẳng thức c1 > x1c5a21, có nghĩa là FUNC (*Qi*) chứa {c1 > \*}. Trong trường hợp này, CryptDB sẽ tiết lộ mã hóa OPE của cột c1 cho máy chủ. Theo giả định, OPE chỉ rò rỉ thứ tự giữa các mục, nghĩa là mục i *>*? Mục J. Tuy nhiên, thông tin này đã được FUNC (*Qi*) cho phép vì máy chủ có thể thay thế c1 bằng mục *i* và ∗ bằng giá trị từ mục *j* in khi đặt câu hỏi toc1 > \*}. Oracle từ tập hợp các câu hỏi được cho phép bởi {

# THỰC HIỆN

Chúng tôi đã triển khai CryptDB trong C ++ trên Postgres 9.0 và sau đó chuyển nó sang MySQL, như đã báo cáo trong §6.4. Chúng tôi đã sử dụng thư viện NTL để thực hiện lý thuyết số trên các số lớn [41] để thực hiện một số giao thức mật mã của chúng tôi. CryptDB là 4700 dòng mã, không kể các dòng trống, thư viện chuẩn, thư viện NTL hoặc mã đánh giá.

Như đã đề cập trước đó, CryptDB không thay đổi nội thất của một

DBMS. Chúng tôi quản lý để thực hiện tất cả các chức năng phía máy chủ với UDF và bảng phía máy chủ. Cái nhìn sâu sắc về lý do tại sao thay đổi mô-đun như vậy là DBMS trên thực tế nằm giữa hai khía cạnh của xử lý truy vấn mà CryptDB phải sửa đổi. Cụ thể, lập kế hoạch và thực hiện truy vấn là giữa phân tích cú pháp truy vấn và các hoạt động cấp thấp trên các mục dữ liệu. Do đó, CryptDB có thể chạy trên bất kỳ DBMS nào tương thích SQL và hỗ trợ UDF, bằng cách viết lại các truy vấn trong giao diện người dùng và thay thế các hoạt động riêng lẻ thông qua UDF.



Hình 5: Thông lượng và độ trễ cho các truy vấn TPC-C không có giao dịch, như một hàm của số lượng máy khách đồng thời.

# ĐÁNH GIÁ THỰC NGHIỆM

Phần này đánh giá ba khía cạnh trong thiết kế của CryptDB và việc triển khai nguyên mẫu của chúng tôi: chi phí hiệu suất, tính di động của CryptDB đối với các máy chủ DBMS khác nhau và tính di động của CryptDB đối với các ứng dụng SQL khác nhau. Kết quả hiển thị trong phần còn lại của phần này cho thấy CryptDB có chi phí thời gian chạy thấp (theo thứ tự chi phí thông lượng 27% và 0*.* Chi phí độ trễ 64 ms cho các truy vấn TPC-C), dễ dàng chuyển sang các máy chủ DBMS mới và không yêu cầu thay đổi mã ứng dụng.

## Kết quả hoạt động tổng thể

Để đo hiệu suất của CryptDB, chúng tôi sử dụng thiết lập thử nghiệm bao gồm một máy chủ có Intel Xeon 3*.CPU* 20 GHz với 4 lõi và 3 GB RAM, và một máy khách có Intel Xeon 1*.* CPU E7310 6 GHz với 16 lõi và RAM 8 GB, trên đó chúng tôi mô phỏng nhiều người dùng. Vì chúng tôi quan tâm đến việc đo lường chi phí biến đổi mật mã của CryptDB, chúng tôi sử dụng khối lượng công việc trong đó toàn bộ tập dữ liệu phù hợp với RAM trên máy chủ DBMS và máy chủ DBMS không bị tắc nghẽn bởi truy cập đĩa (chậm như nhau khi có và không có CryptDB).

Đối với các thử nghiệm hiệu suất của mình, chúng tôi đo lường thông lượng và độ trễ của CryptDB và DBMS Postgres chưa sửa đổi bằng cách sử dụng dấu vết TPCC. Thay vì chỉ chạy điểm chuẩn TPC-C, chúng tôi tạo ra một hỗn hợp các truy vấn OLTP ngẫu nhiên bằng cách thu thập các dấu vết thực thi TPC-C và sau đó có nhiều máy khách trình bày các truy vấn này cho CryptDB.

Cách tiếp cận này cho phép chúng tôi đánh giá chi phí hiệu suất của CryptDB trên nhiều mẫu hình truy cập ngẫu nhiên giống như OLTP.

Một phần đáng kể chi phí CPU của CryptDB chạy các truy vấn TPC-C được che giấu bởi tối ưu hóa bộ nhớ đệm bản mã, bởi vì các truy vấn TPC-C liên tục sử dụng một tập hợp nhỏ các hằng số, có thể hoặc không thể đại diện cho các khối lượng công việc khác. Do đó, chúng tôi cũng báo cáo kết quả cho CryptDB với tối ưu hóa bộ nhớ đệm bản mã bị vô hiệu hóa (được gọi là CryptDB chưa được tối ưu hóa trong kết quả của chúng tôi), đạt được hiệu suất thấp hơn nhưng vẫn hợp lý.

Để đo lường số lượng truy vấn thô mà máy chủ có thể xử lý mỗi giây, trước tiên chúng tôi báo cáo thông lượng cho các truy vấn TPC-C khi các giao dịch không được sử dụng, như thể hiện trong Hình 5. Chúng ta có thể thấy rằng CryptDB phải chịu thông lượng giảm 21% so với máy chủ Postgres chưa được sửa đổi (dựa trên thông lượng cao nhất cho cả hai cấu hình) và CryptDB không được tối ưu hóa phải chịu chi phí 35%. Hơn nữa, do chi phí CPU mã hóa dữ liệu trong giao diện người dùng, CryptDB không được tối ưu hóa yêu cầu nhiều máy khách hơn (chạy trên các lõi khác nhau) để đạt được thông lượng tối đa. Cuối cùng, sự khác biệt về độ trễ giữa CryptDB và máy chủ Postgres chưa sửa đổi là trong vòng 4 ms.

Hình 6 cho thấy thông lượng đạt được cho hỗn hợp truy vấn TPC-C khi các giao dịch được bật. Trong trường hợp này, CryptDB phải chịu hình phạt 27% về thông lượng so với Postgres chưa được sửa đổi, phần lớn là do tranh chấp giao dịch gia tăng do thời gian xử lý phía máy khách lâu hơn và các truy vấn mở rộng. Trong trường hợp

0

20

40

60

80

100

120

0

5

10

15

20

25

30

35

40

Giao dịch/giây

Không. Số lượng khách hàng

Postgres

CryptDB

Hình 6: Thông lượng của các giao dịch TPC-C như một hàm của số lượng khách hàng đồng thời.

CryptDB không được tối ưu hóa, thời gian xử lý phía máy khách cho mỗi truy vấn được tăng lên hơn nữa, dẫn đến xung đột giao dịch nhiều hơn đáng kể và thông lượng tổng thể giảm 70%.

## Truy vấn Microbenchmarks

Để hiểu các nguồn chi phí phát sinh bởi CryptDB, chúng tôi đã kiểm tra thông lượng của các truy vấn riêng lẻ, vì các ứng dụng khác nhau có thể dẫn đến các hỗn hợp truy vấn khác nhau. Đối với mỗi loại truy vấn, chúng tôi đã thu thập các truy vấn tương ứng từ TPC-C và đo độ trễ cho các truy vấn chạy trong CryptDB và trong Postgres chưa được sửa đổi. Kết quả của thí nghiệm này được thể hiện trong Bảng 1. Chúng ta có thể thấy rằng thời gian mã hóa máy khách nói chung là thấp, thêm trung bình là 0*.*34 ms cho truy vấn. CryptDB không được tối ưu hóa cũng có độ trễ thấp ngoại trừ các truy vấn yêu cầu OPE và HOM; độ trễ tổng thể của khách hàng là 7*.*3 ms vẫn có thể được chấp nhận cho một số ứng dụng. Thực tế là độ trễ của máy chủ tương tự đối với CryptDB và Postgres chưa được sửa đổi cho thấy yếu tố mở rộng do mã hóa có tác động vừa phải đến hiệu suất. Tóm lại, tổng thể CryptDB thêm 0*.*Độ trễ 64 ms cho mỗi truy vấn và CryptDB chưa được tối ưu hóa thêm 7*.*6 mili giây.

Hình 7 cho thấy thông lượng cho cùng một hỗn hợp các truy vấn chạy trong CryptDB và Postgres chưa được sửa đổi. Chúng ta có thể thấy rằng đối với sáu loại truy vấn (Chọn đẳng thức, Chọn tham gia, Chọn phạm vi, Xóa, Chèn và Cập nhật bộ), chi phí thông lượng là tối thiểu. Sáu loại truy vấn này tạo thành hầu hết các truy vấn cho cả TPC-C và có khả năng cho nhiều ứng dụng khác, làm cho CryptDB trở thành một lựa chọn tốt. Các phép toán đồng cấu, chẳng hạn như Select sum và Update increment, phát sinh một chi phí đáng kể với CryptDB, do chi phí phía máy chủ để nhân đồng cấu các số mật mã lớn thay vì thêm số nguyên 32 bit. Các ứng dụng sử dụng tổng và gia số nhiều sẽ phải chịu chi phí đáng kể với CryptDB, nhưng các ứng dụng có tỷ lệ phần trăm tổng thấp (chẳng hạn như trong TPC-C), chi phí CryptDB sẽ thấp hơn nhiều.

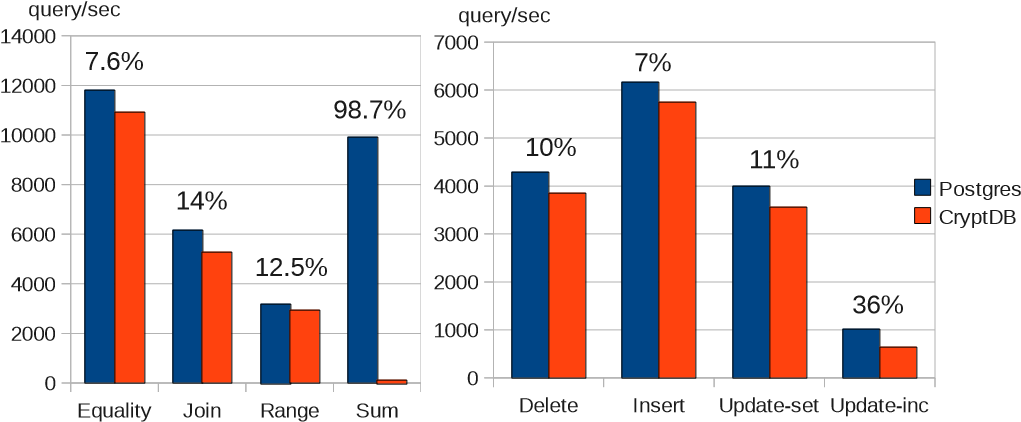
Về mặt lưu trữ, các giao diện người dùng chỉ cần lưu trữ một khóa chính, trạng thái lược đồ và hành tây, cũng như bộ đệm bản mã cho OPE và HOM như một tối ưu hóa. Đối với các truy vấn TPC-C, dung lượng bộ nhớ của quy trình frontend là tối thiểu: 92 kB bộ nhớ, không bao gồm mã và các thư viện được chia sẻ như libc hoặc ít hơn 4 MB bộ nhớ bao gồm mã và thư viện chia sẻ. Nếu giao diện người dùng quyết định lưu vào bộ nhớ cache bản mã cho OPE và HOM, để lưu trữ 100000 giá trị (có lẽ là phổ biến nhất), phải mất < *1*  MB cho OPE và ≈ 12 MB cho HOM.

Về phía máy chủ, CryptDB làm tăng kích thước của cơ sở dữ liệu do mở rộng nhiều củ hành và bản mã. Đối với TPC-C, các

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Truy vấn | CryptDB | | | Postgres |
| Máy chủ | Khách hàng | Bỏ chọn ứng dụng khách. | Máy chủ |
| Chọn đẳng thức | 0.43 | 0.10 | 0.10 | 0.41 |
| Chọn tham gia | 0.72 | 0.27 | 0.27 | 0.63 |
| Chọn phạm vi | 1.2 | 0.40 | 58.2 | 0.99 |
| Chọn tổng | 8.8 | 0.18 | 0.18 | 0.46 |
| Xóa | 1.1 | 0.15 | 0.14 | 1.1 |
| Chèn | 1.0 | 0.34 | 18.6 | 0.99 |
| Bộ cập nhật | 1.2 | 0.17 | 0.17 | 1.1 |
| Gia số cập nhật | 2.0 | 0.71 | 17.7 | 1.8 |
| Chung | 1.4 | 0.34 | 7.3 | 1.1 |

Bảng 1: Số liệu độ trễ tính bằng mili giây. Độ trễ của máy khách là thời gian mã hóa cho CryptDB và CryptDB chưa được tối ưu hóa. Thời gian tổng thể phản ánh độ trễ trung bình cho hỗn hợp

Truy vấn TPC-C. "Nhóm cập nhật" cho biết bản cập nhật trong đó các trường được đặt thành hằng số và "Gia số cập nhật" cho biết bản cập nhật trong đó các trường được tăng lên.



Hình 7: So sánh thông lượng cho các loại truy vấn từ Bảng 1 chạy trong CryptDB và Postgres chưa sửa đổi. Đối với mỗi loại truy vấn, chúng tôi hiển thị phần trăm giảm thông lượng trong CryptDB so với Postgres chưa sửa đổi.

kích thước cơ sở dữ liệu sử dụng Postgres chưa sửa đổi là 135 MB và với CryptDB, nó là 619 MB, lên tới 4*.Tăng* gấp 5 lần. Điều này chủ yếu là do các tập hợp là các số mật mã lớn, mà không có nó là tăng gấp ba lần. Vì dung lượng ổ đĩa tương đối rẻ, chúng tôi không coi việc tăng chi phí lưu trữ là rào cản đáng kể đối với việc áp dụng CryptDB. Chi phí thời gian chạy của CryptDB ít hơn đáng kể so với hệ số mở rộng lưu trữ vì việc đọc một cột (hoặc đánh giá vị ngữ trên cột) chỉ yêu cầu truy cập một trong các củ hành cho cột đó, thay vì mọi củ hành cho cột đó. Hơn nữa, hầu như không có sự mở rộng cho các mục dữ liệu lớn như chuỗi dài hoặc nhị phân. Do đó, một cơ sở dữ liệu lưu trữ văn bản lớn hoặc nhị phân (ví dụ: ảnh) sẽ hầu như không có chi phí lưu trữ.

## Mã hóa dựa trên truy vấn có thể điều chỉnh

Bảo mật có thể điều chỉnh liên quan đến việc giải mã các cột xuống mức hành tây thấp hơn. May mắn thay, việc giải mã như vậy rất nhanh và chỉ cần được thực hiện một lần cho mỗi cột trong suốt vòng đời của hệ thống. Đặc biệt, trong quá trình triển khai của chúng tôi, việc loại bỏ một lớp mã hóa RND yêu cầu giải mã AES, mà máy chủ của chúng tôi có thể thực hiện với tốc độ khoảng 500 MB / giây. Do đó, chi phí loại bỏ một lớp mã hóa củ hành bị tắc nghẽn bởi tốc độ mà máy chủ DBMS có thể sao chép toàn bộ cột dữ liệu trong một bảng duy nhất; Tốc độ giải mã không đáng kể so với băng thông đọc và ghi đĩa.

Bảng 2 cho thấy trạng thái kết quả của các mức mã hóa của các trường khác nhau trong lược đồ của TPC-C sau khi thực hiện các truy vấn TPC-C. Chúng ta có thể thấy rằng 71% các cột vẫn ở RND, có nghĩa là

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RND | DET nhưng không OPE | OPE | Công dụng của HOM | Tất cả |
| 65 | 19 | 8 | 8 | 92 |

Bảng 2: Kết quả trạng thái củ hành của cơ sở dữ liệu cho TPC-C.

rằng không có thông tin rò rỉ về họ. Điều này cho thấy tầm quan trọng của việc tự động điều chỉnh mã hóa dựa trên các truy vấn thay vì chỉ bắt đầu với mã hóa cho phép tất cả các hoạt động. Chỉ đối với OPE, 2 cột có truy vấn phạm vi, 6 cột còn lại được giải mã thành OPE do ORDER BY. Người ta có thể tránh giải mã như vậy bằng cách đơn giản là đặt hàng các mục trên giao diện người dùng.

## Tính di động của máy chủ

Để chứng minh tính di động của CryptDB, chúng tôi đã chuyển CryptDB sang MySQL. Chúng tôi chỉ phải thay đổi và thêm tổng cộng 86 dòng mã. Mã này chủ yếu là mã kết nối, cho phép giao diện người dùng CryptDB kết nối với máy chủ MySQL, một định dạng mới cho khai báo UDF (mặc dù nội dung giống nhau) và xử lý thông tin được gửi và nhận khác nhau từ máy chủ. Tuy nhiên, chúng tôi không thay đổi bất kỳ mã DBMS nào trong máy chủ MySQL và không thay đổi bất kỳ mã nào cho logic của CryptDB trong giao diện người dùng. Điều này cho thấy CryptDB có thể dễ dàng chuyển sang các máy chủ DBMS khác hỗ trợ UDF.

## Tính di động của ứng dụng

Để chứng minh tính di động của các ứng dụng hiện có để:

CryptDB, chúng tôi đã chạy một số ứng dụng trên CryptDB, bao gồm TPCC và hai phiên bản khác nhau của ứng dụng web tuyển sinh sau đại học của một tổ chức học thuật, một được viết bằng các truy vấn SQL viết tay và một được viết bằng hệ thống Django ORM [12] cho Python. CryptDB có thể hỗ trợ tất cả các truy vấn SQL từ các ứng dụng này mà không cần phải sửa đổi các truy vấn hoặc ứng dụng theo bất kỳ cách nào.

# CÔNG VIỆC LIÊN QUAN

Ở cấp độ cao, đóng góp chính của CryptDB so với công việc trước đây là một cách tiếp cận mới thực tế để đảm bảo quyền riêng tư dữ liệu trong DBMS. Theo hiểu biết của chúng tôi, CryptDB là hệ thống riêng đầu tiên hỗ trợ tất cả các toán tử được sử dụng phổ biến trong SQL, thực hiện hầu như tất cả các xử lý truy vấn trên máy chủ, hoạt động mà không cần sửa đổi nội bộ của các cơ sở mã DBMS quan hệ hiện có hoặc các ứng dụng máy khách và chạy ở mức suy giảm hiệu suất khá khiêm tốn.

Chúng tôi chia công việc liên quan thành mật mã ứng dụng, cung cấp các công cụ để xử lý dữ liệu được mã hóa, nhưng không phải là giải pháp hệ thống toàn diện, phương pháp tiếp cận hệ thống được thực hiện trong bối cảnh thiết kế được triển khai và phương pháp lý thuyết.

*Mật mã ứng dụng.*

Trong vài năm qua, các nhà nghiên cứu đã phát triển các công cụ mật mã để tìm kiếm từ khóa trên văn bản được mã hóa [4, 43] và một số nhà nghiên cứu đã đề xuất sử dụng các công cụ này để xử lý các truy vấn SQL trên dữ liệu được mã hóa [2, 5, 14, 51]. Các cách tiếp cận này là một cách cắt giảm đầu tiên tốt cho vấn đề, nhưng không đầy đủ theo những cách thực chất: chúng không hỗ trợ nhiều truy vấn SQL cơ bản, chủ yếu chỉ cung cấp so sánh bình đẳng, hầu hết chúng yêu cầu xử lý truy vấn phía máy khách đáng kể, yêu cầu thay đổi nội bộ của DBMS; Hầu hết đều quá kém hiệu quả, ví dụ: yêu cầu người dùng xây dựng và duy trì các chỉ mục trên dữ liệu tại máy chủ hoặc thực hiện quét tuần tự cho mọi lựa chọn / tìm kiếm; Cuối cùng, nhiều người vẫn ở cấp độ của các giao thức mật mã và không xây dựng và chứng minh một hệ thống. Tuy nhiên, những cách tiếp cận này rất hữu ích cho việc tìm kiếm văn bản riêng tư và chúng tôi sử dụng một phương pháp tương tự cho "ILIKE" với các phương pháp trong [1, 43]; tuy nhiên, chúng không tạo thành DBMS bảo vệ quyền riêng tư hoàn chỉnh hoặc hiệu quả.

Amanatidis et al. [1], hơi khác so với các sơ đồ trên, cho phép máy chủ xây dựng các chỉ mục, nhưng quyền riêng tư của chúng thấp hơn đáng kể so với CryptDB: chúng tiết lộ *trước tất cả các giá trị lặp lại và tất cả các* phạm vi chung (cùng giá trị tiền tố) trong toàn bộ cơ sở dữ liệu (trên tất cả các cột và hàng). Chìa khóa bảo mật của chúng tôi là mã hóa có thể điều chỉnh động dựa trên các truy vấn: các trường không bao giờ được sử dụng trong các truy vấn phạm vi hoặc đẳng thức sẽ vẫn được mã hóa với sơ đồ mã hóa mạnh nhất. Cuối cùng, chúng không hỗ trợ tất cả các truy vấn khác ngoài phạm vi đẳng thức và dựa trên tiền tố và không xây dựng hệ thống.

Ngoài ra, quyền riêng tư được cung cấp bởi các chương trình an toàn nhất trong số các chương trình được phát triển trước đây này gần giống với CryptDB. Ví dụ, trong công việc của Yang et al. [51], sau khi một truy vấn đã được thực hiện, máy chủ chỉ biết sự lặp lại của hằng số trong bộ lọc lựa chọn và không biết tất cả các lần lặp lại dữ liệu trong một cột như trong CryptDB. Do đó, sau một truy vấn đẳng thức, giao thức của họ an toàn hơn. Tuy nhiên, sau một vài truy vấn khác nhau với cùng một cấu trúc, nhưng các hằng số khác nhau, CryptDB tiết lộ nhiều thông tin như cách tiếp cận của họ. Đây là những công cụ cụ thể và không cho phép máy chủ thực hiện các yêu cầu khác trong DBMS: truy vấn phạm vi chung, cập nhật toàn bộ cột hoặc phạm vi bộ dữ liệu (ví dụ: gia số), tổng hợp, một số không thể hỗ trợ nối.

Ngoài ra còn có công việc cho phép máy chủ xây dựng các chỉ mục an toàn [18] trên dữ liệu được mã hóa mà không cần truy cập vào dữ liệu. Những cách tiếp cận này đòi hỏi những thay đổi đáng kể đối với DBMS và sẽ không di động. Hơn nữa, do các công cụ mật mã phức tạp, chúng chậm hơn. Trong trường hợp của chúng tôi, máy chủ xây dựng các chỉ mục một cách tự nhiên như trên các số điển hình (dài hơn). Ge và Zdonik [15] cho phép so sánh và thiết kế chỉ mục cho một cửa hàng cột.

Đã có nghiên cứu xử lý các truy vấn trên XML được mã hóa [20, 46]. Công việc này cung cấp ngữ nghĩa bảo mật hữu ích cho dữ liệu XML, nhưng yêu cầu viết lại DBMS cồng kềnh. Hơn nữa, kết quả gửi cho khách hàng là một siêu tập hợp của kết quả thực, yêu cầu khách hàng thực hiện xử lý hậu kỳ bổ sung. Họ cũng không thảo luận về cập nhật và chèn.

*Phương pháp tiếp cận hệ thống.*

Trong khi các đề xuất ban đầu cố gắng cho phép xử lý SQL trên dữ liệu được mã hóa, các cơ chế bảo mật của chúng mang tính phỏng đoán mà không có sự đảm bảo chính thức, yêu cầu viết lại đáng kể thiết kế DBMS, dựa vào xử lý phía máy khách đáng kể và không hỗ trợ một loạt các truy vấn SQL.

Trong [19], Hacigumus et al. chia miền của các giá trị có thể có cho mỗi cột trong các phân vùng. Mỗi phân vùng có một số (ngẫu nhiên hoặc bảo toàn thứ tự) và mỗi giá trị được thay thế bằng số lượng phân vùng. Các tuple thực tế được lưu trữ mã hóa. Bằng cách nhóm các phần tử trong phân vùng, quyền riêng tư có thể bị xâm phạm vì một thực thể không đáng tin cậy có thể biết rõ phần tử nào gần nhau về giá trị. Càng nhiều phân vùng, máy chủ càng thực hiện nhiều công việc hữu ích, nhưng máy khách càng có ít quyền riêng tư. Với cơ chế mã hóa có thể điều chỉnh động của chúng tôi, chúng tôi không tiết lộ mối quan hệ cho các cột không được sử dụng trong bộ lọc.

Ozsoyoglu et al. [34] sử dụng các chức năng do người dùng xác định và không yêu cầu thay đổi phần mềm DBMS. Tuy nhiên, cách tiếp cận của họ chỉ áp dụng cho số nguyên, không hỗ trợ nối (phải được xử lý tại máy khách), không thảo luận về cập nhật và chèn. Điều quan trọng, chúng  *mã hóa các trường với chức năng mã hóa bảo toàn* khoảng cách mà a *− b* =  *E (a*) *-* E(b*)*. Lược đồ như vậy không an toàn vì a đối với mã hóa *E* (*a*) làm rò rỉ kiến thức đơn thuần về một giá trị giải mã tất cả các giá trị được mã hóa khác.

Damiani et al. [10] cung cấp một giải pháp để xử lý các truy vấn phạm vi trên dữ liệu được mã hóa. Mỗi phần tử trong một bộ dữ liệu được mã hóa và băm vào một số lượng nhỏ các vùng lưu trữ để cải thiện quyền riêng tư. Các truy vấn phạm vi có thể được tính toán bằng cách sử dụng cây B +, nhưng chúng được thực hiện bởi máy khách đáng tin cậy cần đi qua cây B + bằng cách thực hiện tuần tự các truy vấn tại máy chủ. Công việc như vậy không hỗ trợ nối, tổng hợp hoặc tìm kiếm chuỗi.

Ciriani et. al [8] đề xuất một cách tiếp cận mới để bảo mật: thay thế mã hóa dữ liệu bằng phân mảnh; Chúng lưu trữ một phần dữ liệu tại máy khách đáng tin cậy (ví dụ: các cột nhạy cảm hoặc quan hệ giữa các cột) và phần còn lại của dữ liệu không được mã hóa tại máy chủ không đáng tin cậy, do đó tránh mã hóa hoàn toàn. Tuy nhiên, mỗi khách hàng (đáng tin cậy) phải lưu trữ một lượng lớn dữ liệu cục bộ và phải thực hiện xử lý truy vấn bất cứ khi nào dữ liệu nhạy cảm có liên quan đến truy vấn.

Chow et al. [7] yêu cầu sự hiện diện của hai bên bổ sung, một bộ ngẫu nhiên và một công cụ truy vấn, được giả định là không giao tiếp; hơn nữa, dữ liệu được lưu trữ tại nhiều DBMS không tin tưởng lẫn nhau. Tính bảo mật của các giao thức của họ xoay quanh các giả định bảo mật như vậy và không có sự thông đồng nào xảy ra; Một thiết lập như vậy không phải lúc nào cũng có thể, và các giải pháp mà không có giả định tin cậy mạnh mẽ như vậy có thể là mong muốn.

*Phương pháp tiếp cận lý thuyết.*

Các giải pháp lý thuyết hứa hẹn một mức độ bảo mật cao, tuy nhiên, chúng rất không thực tế. Công việc gần đây [16] dựa trên mã hóa đồng cấu hoàn toàn [17] cho phép một bên không đáng tin cậy thực hiện bất kỳ tính toán chung nào trên dữ liệu được mã hóa (tính toán như vậy có thể là xử lý truy vấn) mà không làm rò rỉ ngay cả các mẫu truy cập vào dữ liệu. Thật không may, ví dụ, thực hiện tìm kiếm chuỗi đơn giản bằng cách sử dụng mã hóa đồng cấu chậm hơn khoảng một nghìn tỷ lần so với không mã hóa [9].

Làm việc trong PIR [33] cho phép người dùng yêu cầu một bộ dữ liệu từ cơ sở dữ liệu mà không cần máy chủ tìm hiểu những gì tuple đã được yêu cầu. Mặc dù chúng cung cấp bảo mật tuyệt vời, nhưng các cách tiếp cận như vậy rất không khả thi bởi vì, đối với mỗi bộ dữ liệu được yêu cầu, máy chủ phải quét toàn bộ cơ sở dữ liệu.

*Công việc khác.*

Đã có một khối lượng công việc quan trọng về bảo vệ quyền riêng tư phân tán bảo tồn tích hợp, tổng hợp và khai thác dữ liệu: xây dựng cây quyết định [27], quy tắc liên kết máy tính, phân loại và phân cụm [23, 24, 45, 50], tổng hợp dữ liệu thuê ngoài [48] và [44]. Những tác phẩm này cung cấp một bộ công cụ bảo vệ quyền riêng tư phong phú và hữu ích cho các mục đích khác với CryptDB và chúng có thể được sử dụng cùng với CryptDB.

Làm việc trên quyền riêng tư khác biệt [13, 29, 38], được giới thiệu bởi Dwork và được tăng cường bởi Miklau [30], cũng như quyền riêng tư trong cơ sở dữ liệu thống kê, cho phép một máy chủ đáng tin cậy quyết định câu trả lời nào sẽ phát hành hoặc cách làm xáo trộn câu trả lời cho các truy vấn tổng hợp để tránh rò rỉ thông tin về bất kỳ bản ghi cụ thể nào trong cơ sở dữ liệu. Công việc như vậy có một mục tiêu và mô hình khác với CryptDB; trên thực tế, người ta có thể thêm quyền riêng tư khác biệt vào giao diện người dùng trong CryptDB để chỉ cung cấp cho người dùng các câu trả lời bảo vệ quyền riêng tư.

# KẾT THÚC

Bài viết này trình bày CryptDB, một hệ thống thực tế và mới lạ để đảm bảo quyền riêng tư dữ liệu trên máy chủ SQL DBMS không đáng tin cậy. CryptDB sử dụng ba ý tưởng mới để đạt được mục tiêu của mình: *chiến lược mã hóa nhận biết SQL*, mã hóa *dựa trên truy vấn có thể điều chỉnh* và *mã hóa hành tây*. Là một phần của chiến lược mã hóa nhận biết SQL của CryptDB, chúng tôi đề xuất tối ưu hóa cho các kỹ thuật mật mã hiện có, cũng như một cơ chế mật mã mới cho các tham gia riêng tư. Nguyên mẫu CryptDB của chúng tôi không yêu cầu thay đổi mã ứng dụng hoặc mã máy chủ DBMS và sử dụng các chức năng do người dùng xác định để thực hiện các hoạt động mật mã bên trong công cụ DBMS hiện có, bao gồm cả Postgres và MySQL. Trong khối lượng công việc TPC-C, nguyên mẫu của chúng tôi phải chịu một

Giảm 27% thông lượng so với DBMS không được mã hóa, làm cho CryptDB trở thành một lựa chọn thiết thực cho dữ liệu nhạy cảm về quyền riêng tư.

# Lời cảm ơn

Các tác giả xin cảm ơn Carlo Curino và Sam Madden vì những phản hồi hữu ích. Carlo Curino và Evan Jones đã cung cấp dấu vết được sử dụng trong đánh giá TPC-C của chúng tôi. Eugene Wu và Alvin Cheung cũng đưa ra những lời khuyên hữu ích. Công việc này được hỗ trợ bởi NSF (CNS0716273) và Google.

# Tham khảo

1. G. Amanatidis, A. Boldyreva và A. O'Neill. Các sơ đồ an toàn có thể chứng minh để hỗ trợ truy vấn cơ bản trong cơ sở dữ liệu thuê ngoài. Trong *Kỷ yếu của Hội nghị làm việc IFIP WG 11.3 thường niên lần thứ 21 về bảo mật cơ sở dữ liệu và ứng dụng,* trang 14–30, Redondo Beach, CA, tháng 7 năm 2007.
2. F. Bao, R. H. Deng, X. Ding và Y. Yang. Truy vấn riêng tư trên dữ liệu được mã hóa trong cài đặt nhiều người dùng. Trong *Kỷ yếu của Hội nghị Quốc tế lần thứ 4 về Thực hành và Kinh nghiệm An toàn Thông tin,* trang 71–85, Sydney, Úc, 2008.
3. A. Boldyreva, N. Chenette, Y. Lee và A. O'Neill. Bảo quản thứ tự mã hóa đối xứng. Trong *Kỷ yếu của Hội nghị quốc tế thường niên lần thứ 28 về những tiến bộ trong mật mã*, trang 224–241, Cologne, Đức, 2009.
4. D. Boneh và B. Waters. Truy vấn kết hợp, tập con và phạm vi trên dữ liệu được mã hóa. Trong *Kỷ yếu của Hội nghị lần thứ 4 về Lý thuyết Mật mã*, trang 535–554, Amsterdam, Hà Lan, 2007.
5. Y.-C. Chang và M. Mitzenmacher. Bảo vệ quyền riêng tư khi tìm kiếm từ khóa trên dữ liệu được mã hóa từ xa. Trong *Kỷ yếu của Hội nghị Mật mã học và An ninh mạng Ứng dụng lần thứ 3,* New York, NY, tháng 6 năm 2005.
6. A. Trần. GCreep: Kỹ sư Google rình rập thanh thiếu niên, theo dõi các cuộc trò chuyện. *Gawker*, tháng 9 năm 2010. http://gawker.com/ 5637234/.
7. S. S. M. Chow, JH Lee và L. Subramanian. Mô hình tính toán hai bên cho các truy vấn bảo vệ quyền riêng tư trên cơ sở dữ liệu phân tán. Trong *Kỷ yếu của Hội nghị chuyên đề an ninh mạng và hệ thống phân tán lần thứ 16*, 2009.
8. V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi và P. Samarati. Giữ một vài: Thuê ngoài dữ liệu trong khi vẫn duy trì tính bảo mật. Trong *Kỷ yếu của Hội nghị chuyên đề châu Âu lần thứ 14 về nghiên cứu bảo mật máy tính*, 2009.
9. M. Cooney. IBM chào mời đổi mới mã hóa; Công nghệ mới thực hiện tính toán trên dữ liệu được mã hóa mà không cần giải mã nó. *Thế giới máy tính*, 2009.
10. E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi và P. Samarati. Cân bằng tính bảo mật và hiệu quả trong các DBMS quan hệ không đáng tin cậy. Trong *Kỷ yếu của Hội nghị ACM lần thứ 10 về An ninh Máy tính và Truyền thông,* Washington, DC, tháng 10 năm 2003.
11. A. Desai. Các mô hình mới để xây dựng các sơ đồ mã hóa đối xứng an toàn chống lại cuộc tấn công bản mã đã chọn. Trong *Kỷ yếu của Hội nghị quốc tế thường niên lần thứ 20 về những tiến bộ trong mật mã học,* trang 394–412, tháng Tám năm 2000.
12. Quỹ phần mềm Django. Django: Khung web dành cho những người cầu toàn có thời hạn. http://www. djangoproject.com/, tháng 10 năm 2010.
13. C. Làm việc. Quyền riêng tư khác biệt: một cuộc khảo sát về kết quả. Trong *Kỷ yếu của Hội nghị Quốc tế lần thứ 5 về Lý thuyết và Ứng dụng các Mô hình Tính toán (TAMC'08),* trang 1–19, Tây An, Trung Quốc, 2008.
14. S. Evdokimov và O. Guenther. Kỹ thuật mã hóa để thuê ngoài cơ sở dữ liệu an toàn. Cryptology ePrint Archive, Báo cáo 2007/335, 2007. http://eprint.iacr.org/.
15. T. Ge và S. Zdonik. Mã hóa nhanh, an toàn để lập chỉ mục trong DBMS hướng cột. *ICDE,* 2007.
16. R. Gennaro, C. Gentry và B. Parno. Điện toán có thể xác minh không tương tác: Thuê ngoài tính toán cho những người lao động không đáng tin cậy. Trong *Kỷ yếu của STOC*, 2010.
17. C. Gentry. Mã hóa hoàn toàn đồng cấu bằng cách sử dụng mạng lý tưởng. Trong *Kỷ yếu của Hội nghị chuyên đề ACM thường niên lần thứ 41 về Lý thuyết Máy tính*, trang 169–178, New York, NY, Hoa Kỳ, 2009. ACM.
18. E.-J. Goh. Chỉ mục bảo mật. Cryptology ePrint Archive, Báo cáo 2003/216, 2003. http://eprint.iacr.org/2003/ 216/.
19. H. Hacigumus, B. Iyer, C. Li và S. Mehrotra. Thực thi SQL trên dữ liệu được mã hóa trong mô hình cơ sở dữ liệu-dịch vụ-nhà cung cấp. Trong *Kỷ yếu của Hội nghị Quốc tế ACM SIGMOD 2002 về Quản lý Dữ liệu,* Madison, WI, tháng Sáu năm 2002.
20. R. C. Jammalamadaka và S. Mehrotra. Truy vấn tài liệu XML được mã hóa. Trong *Kỷ yếu của Quốc tế 10*

*Hội nghị chuyên đề về ứng dụng và kỹ thuật cơ sở dữ liệu*, trang 129–136, Washington, DC, Hoa Kỳ, 2006. Hiệp hội Máy tính IEEE.

1. V. Kachitvichyanukul và B. W. Schmeiser. Thuật toán 668: H2pec: lấy mẫu từ phân bố siêu hình học. *ACM Trans. Toán học. Softw.*, 14(4):397–398, 1988.
2. M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang và K. Fu. Plutus: Chia sẻ tệp an toàn có thể mở rộng trên bộ nhớ không đáng tin cậy.

Trong *Kỷ yếu của Hội nghị USENIX lần thứ 2 về Công nghệ Tệp và Lưu trữ,* trang 29–42, Berkeley, CA, Hoa Kỳ, 2003. Hiệp hội USENIX.

1. M. Kantarcioglu và C. Clifton. Các vấn đề bảo mật trong việc truy vấn dữ liệu được mã hóa. Báo cáo kỹ thuật CSD TR 04-013, Đại học Purdue, Khoa Khoa học Máy tính, 2004.
2. M. Kantarcioglu và C. Clifton. Các vấn đề bảo mật trong việc truy vấn dữ liệu được mã hóa. Trong *Kỷ yếu của Hội nghị làm việc IFIP WG 11.3 thường niên lần thứ 19 về bảo mật cơ sở dữ liệu và ứng dụng*, 2005.
3. F. Li, M. Hadjieleftheriou, G. Kollios và L. Reyzin. Cấu trúc chỉ mục được xác thực cho các truy vấn tổng hợp trong cơ sở dữ liệu thuê ngoài. *Báo cáo kỹ thuật BUCS-TR-2006-011*, 2006.
4. J. Li, M. Krohn, D. Mazières và D. Shasha. Bảo mật kho dữ liệu không đáng tin cậy (SUNDR). Trong *Kỷ yếu của Hội nghị chuyên đề lần thứ 6 về Thiết kế và Triển khai Hệ điều hành,* trang 91–106, San Francisco, CA, tháng Mười Hai năm 2004.
5. Y. Lindell và B. Pinkas. Bảo vệ quyền riêng tư khi khai thác dữ liệu. Trong *Tạp chí Mật mã*, trang 36–54. Springer-Verlag, 2000.
6. V. Maine và các cộng sự. Luật Bảo mật Dữ liệu Massachusetts 201 CMR 17.00, 2010.
7. F. McSherry. Truy vấn tích hợp quyền riêng tư. Trong *Kỷ yếu của Hội nghị Quốc tế ACM SIGMOD 2009 về Quản lý Dữ liệu,* Providence, RI, tháng Sáu – tháng Bảy năm 2009.
8. G. Miklau. Tăng độ chính xác của các truy vấn riêng tư khác biệt thông qua tính nhất quán. Trong *kỷ yếu của Hội nghị quốc tế lần thứ 36 về cơ sở dữ liệu rất lớn,* Singapore, tháng 9 năm 2010.
9. Cơ quan An ninh Quốc gia. Trường hợp mật mã đường cong elip. http://www.nsa.gov/business/ chương trình/elliptic\_curve.shtml, 2009.
10. V. H. Nguyễn, T. K. Đăng, N. T. Sơn và J. Kung. Xác minh đảm bảo truy vấn cho cơ sở dữ liệu XML thuê ngoài động. *Hội nghị về tính sẵn có, độ tin cậy và an ninh,* 2007.
11. R. Ostrovsky và W. Skeith. Một cuộc khảo sát về truy xuất thông tin cá nhân cơ sở dữ liệu đơn: Kỹ thuật và ứng dụng. Trong *Proceedings of the Public Key Cryptography*, trang 393–411, 2007.
12. G. Ozsoyoglu, D. A. Singer, và S. S. Chung. Cơ sở dữ liệu chống giả mạo: Truy vấn cơ sở dữ liệu được mã hóa. Trong *Kỷ yếu của Hội nghị làm việc IFIP WG 11.3 thường niên lần thứ 17 về bảo mật cơ sở dữ liệu và ứng dụng*, trang 4–6, Estes Park, CO, 2003.
13. P. Paillier. Các hệ thống mật mã khóa công khai dựa trên các lớp dư lượng bằng tổng hợp. Trong *Những tiến bộ trong mật mã học - EUROCRYPT '99*, trang 223–238. Springer-Verlag, 1999.
14. R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang và L. Zhuang. Bật bảo mật trong SLA lưu trữ đám mây với CloudProof. *Báo cáo Kỹ thuật của Microsoft*, 2009.
15. Quyền riêng tư Clearinghouse. Trình tự thời gian của các vi phạm dữ liệu. http://www.privacyrights.org/data-breach.
16. I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, và E. Witchel. Airavat: Bảo mật và quyền riêng tư cho mapreduce. Trong *Kỷ yếu của Hội nghị chuyên đề lần thứ 7 về Thiết kế và Triển khai Hệ thống Nối mạng*, San Jose, CA, Tháng Tư 2010.
17. H. Shacham, N. Modadugu, và D. Boneh. Sirius: Bảo mật bộ nhớ không đáng tin cậy từ xa. Trong *Kỷ yếu của Hội nghị chuyên đề an ninh mạng và hệ thống phân tán lần thứ 10*, 2003.
18. E. Shi, J. Bethencourt, H. Chan, D. Song và A. Perrig. Truy vấn phạm vi đa chiều trên dữ liệu được mã hóa. Trong *Kỷ yếu của Hội nghị chuyên đề IEEE lần thứ 28 về An ninh và Quyền riêng tư*, Oakland, CA, tháng 5 năm 2007.
19. V. Shoup. NTL: Một thư viện để làm lý thuyết số. http:

www.shoup.net/ntl/, tháng 8 năm 2009.

1. R. Sion. Đảm bảo thực hiện truy vấn cho cơ sở dữ liệu thuê ngoài.

Trong *Kỷ yếu của Hội nghị quốc tế lần thứ 31 về các cơ sở dữ liệu rất lớn,* các trang 601–612, Trondheim, Na Uy, tháng Tám–tháng Chín năm 2005.

1. D. X. Song, D. Wagner, S. David, và A. Perrig. Kỹ thuật thực hành để tìm kiếm trên dữ liệu được mã hóa. Trong *Kỷ yếu của Hội nghị chuyên đề IEEE lần thứ 21 về An ninh và Quyền riêng tư*, Oakland, CA, tháng 5 năm 2000.
2. B. Thompson, S. Haber, W. G. Horne, T. S, và D. Yao.

Tính toán bảo vệ quyền riêng tư và xác minh các truy vấn tổng hợp trên cơ sở dữ liệu thuê ngoài. Báo cáo kỹ thuật HPL-2009119, HP Labs, 2009.

1. J. Vaidya và C. Clifton. Khai thác quy tắc liên kết bảo vệ quyền riêng tư trong dữ liệu được phân vùng theo chiều dọc. Trong *Kỷ yếu của Hội nghị Quốc tế ACM SIGKDD lần thứ 8 về Khám phá tri thức và Khai thác dữ liệu,* trang 639–644, 2002.
2. H. W. Vương. L.V.S.: Đánh giá truy vấn an toàn hiệu quả trên cơ sở dữ liệu XML được mã hóa. Trong *kỷ yếu của Hội nghị quốc tế lần thứ 32 về cơ sở dữ liệu rất lớn,* Seoul, Hàn Quốc, tháng 9 năm 2006.
3. Cập nhật tiến độ triển khai beta riêng tư của Xeround, Inc. và một số chỉ số. http://blog.xeround.com/2010/09/27/ private-beta-rollout-progress-update.
4. L. Xiong, S. Chitti và L. Liu. Bảo vệ quyền riêng tư dữ liệu cho các dịch vụ tổng hợp dữ liệu thuê ngoài. Báo cáo kỹ thuật, Đại học Emory, Khoa Toán học và Khoa học Máy tính, 2007.
5. Y. Yang, D. Papadias, S. Papadopoulos và P. Kalnis. Xử lý tham gia được xác thực trong cơ sở dữ liệu thuê ngoài. Trong *Kỷ yếu của Hội nghị Quốc tế ACM SIGMOD 2009 về Quản lý Dữ liệu,* trang 5–18, Providence, RI, tháng 6–tháng 7 năm 2009.
6. Z. Yang, S. Zhong và R. N. Wright. Phân loại dữ liệu khách hàng bảo vệ quyền riêng tư mà không làm mất độ chính xác. Trong *Kỷ yếu của Hội nghị Quốc tế SIAM lần thứ 5 về Khai thác Dữ liệu,* trang 21–23, Newport Beach, CA, 2005.
7. Z. Yang, S. Zhong và R. N. Wright. Truy vấn bảo vệ quyền riêng tư trên dữ liệu được mã hóa. Trong *Kỷ yếu của Hội nghị chuyên đề châu Âu lần thứ 11 về nghiên cứu bảo mật máy tính*, 2006.

# PHỤ LỤC

Chúng tôi mô tả chi tiết hơn FUNC (*Q*) (được định nghĩa trong § 2.1), cho bất kỳ *Q* nào một truy vấn SQL. Quan hệ là một biểu thức của dạng {x OP y}, trong đó OP có thể là bất kỳ phép toán nào (chẳng hạn như =*,>,<,*≥,*≤*, ILIKEx và *y*) có thể được thực hiện giữa các cột và / hoặc hằng số. là tên của các cột trong cơ sở dữ liệu hoặc ký tự đại diện ∗ (nghĩa là bất kỳ giá trị hằng số nào). Khởi tạo mối quan hệ là một biểu thức  trong đó mỗi toán hạng cột được thay thế bằng chỉ mục của một mục trong cột và ∗table1.c1 = constis được thay thế bằng một số giá trị. Ví dụ,} là {item i của một khởi tạo của {

bảng1.c1 = x1c5a21}. Việc đánh giá một sự khởi tạo là một giá trị boolean: true nếu sự khởi tạo của mối quan hệ là đúng và sai nếu không.

Sau đó, FUNC (*Q*) là tập hợp tất cả các mối quan hệ mà có sự khởi tạo mà máy chủ cần biết để thực hiện xử lý SQL cho *Q*.

Ví dụ: cho Q: CHỌN MAX(table1.c1) TỪ table1, table2 WHERE table1.c2 = table2.c1

{VÀ table2.c2 = x1c5a21table1.c1 > table1.c1} . (forThen,MAX),FUNC{table1.c(*Q*) = 2 ={ một vị ngữ của formtable2.c1}, {table2.cc1 = c22 = , điều này giới thiệu {\*} }. Đối với các phép nối, nếu chúng ta cóc1 = c2},

{c1 = \*} và {c2 = \*} trong FUNC(). Chèn và tổng hợp không làm tăng FUNC (). Cập nhật theo gia số cũng không làm tăng FUNC (); Cập nhật bằng cách đặt một giá trị như SET c1 = x1c5a21 thêm c1 = *∗* vào FUNC; Xóa thêm mối quan hệ tương ứng với bộ lọc của chúng như đã thảo luận ở trên.

1. CryptDB không bao giờ cung cấp cho máy chủ các khóa củ hành để giải mã dữ liệu thành bản rõ. [↑](#footnote-ref-1)
2. Trên thực tế, CryptDB cũng sẽ hoạt động với bất kỳ máy chủ DBMS quan hệ hàng hóa chưa sửa đổi nào khác. [↑](#footnote-ref-2)
3. Nếu truy vấn SELECT đầu tiên trả về nhiều kết quả, giao diện người dùng sẽ tính ánh xạ các giá trị lương được mã hóa cũ và các giá trị lương được mã hóa mới tương ứng và sử dụng UDF trong truy vấn CẬP NHẬT thứ hai để cập nhật cột lương theo ánh xạ này. [↑](#footnote-ref-3)