**Queryable Encryption**

**(Mã hóa có thể truy vấn)**

1. Tổng quan

“Queryable Encryption” (Mã hóa có thể truy vấn) là một tính năng của MongoDB cho phép thực hiện các công việc sau:

* Mã hóa các trường dữ liệu nhạy cảm từ phía máy khách: Dữ liệu nhạy cảm được mã hóa ngay tại máy khách trước khi được gửi qua mạng đến MongoDB.
* Lưu trữ các trường dữ liệu nhạy cảm dưới dạng dữ liệu được mã hóa hoàn toàn ngẫu nhiên trên máy chủ cơ sở dữ liệu: Dữ liệu được lưu trữ trên máy chủ MongoDB dưới dạng dữ liệu đã được mã hóa.
* Chạy các truy vấn biểu cảm trên dữ liệu đã được mã hóa: Các truy vấn có thể được thực hiện trên dữ liệu đã được mã hóa mà không cần giải mã.
* Các công việc này đều được hoàn thành mà máy chủ không cần biết bất kỳ thông tin gì về dữ liệu đang được xử lý. Dữ liệu nhạy cảm được mã hóa trong suốt quá trình vận hành - trong quá trình truyền, tại chỗ, trong quá trình sử dụng, trong nhật ký, và sao lưu - và chỉ được giải mã ở phía máy khách, vì chỉ mới có quyền truy cập vào các khóa mã hóa.

“Queryable Encryption” giới thiệu một lược đồ mã hóa có thể tìm kiếm nhanh, đầu tiên trong ngành, được phát triển bởi các nhà tiên phong trong tìm kiếm mã hóa. Tính năng này hỗ trợ các truy vấn đẳng thức, với các loại truy vấn bổ sung như phạm vi, tiền tố, hậu tố, và chuỗi con được lên kế hoạch cho các phiên bản phát hành trong tương lai.

Có thể thiết lập “Queryable Encryption” bằng cách sử dụng các cơ chế sau:

* Mã hóa tự động: Cho phép thực hiện các hoạt động đọc và ghi đã mã hóa mà không cần phải viết mã để chỉ định cách mã hóa các trường.
* Mã hóa rõ ràng: Cho phép thực hiện các hoạt động đọc và ghi đã mã hóa thông qua thư viện mã hóa của trình điều khiển MongoDB. phải chỉ định logic cho mã hóa với thư viện này trong suốt ứng dụng của .

1. Mục tiêu thiết kế QE

* Truy vấn biểu thức: Mục tiêu đầu tiên của QE là hỗ trợ các truy vấn biểu thức. Trong khi CSFLE chỉ hỗ trợ các toán tử so sánh bằng ($eq), QE được thiết kế để mở rộng hỗ trợ cho nhiều toán tử khác như $gt (lớn hơn), $gte (lớn hơn hoặc bằng), $lt (nhỏ hơn) và $lte (nhỏ hơn hoặc bằng). QE cũng được thiết kế để hỗ trợ các toán tử phạm vi, tiền tố, hậu tố và chuỗi con.

Ví dụ : Truy vấn biểu thức: Giả sử muốn tìm tất cả các bệnh nhân có tuổi từ 30 đến 40. Với QE, có thể thực hiện truy vấn này ngay cả khi trường age đã được mã hóa:

db.patients.find({ age: { $gte: 30, $lte: 40 } })

* Hiệu suất: Mục tiêu thứ hai của QE là đảm bảo hiệu suất. Một cơ sở dữ liệu được mã hóa bằng CSFLE gần như không gây ra bất kỳ overhead nào so với cơ sở dữ liệu gốc. QE được thiết kế để có hiệu suất truy vấn chậm hơn khoảng 10 lần so với truy vấn gốc.
* Không tương tác: Mục tiêu thứ ba của QE là giữ cho các hoạt động chuẩn của MongoDB là một vòng đơn và không yêu cầu sự tương tác giữa khách hàng và máy chủ.

Ví dụ : QE không yêu cầu bất kỳ sự tương tác nào giữa khách hàng và máy chủ để thực hiện các hoạt động trên cơ sở dữ liệu. Điều này có nghĩa là có thể thực hiện các truy vấn và thao tác dữ liệu như bình thường mà không cần phải quan tâm đến việc dữ liệu đã được mã hóa.

* Không trạng thái: Mục tiêu thứ tư của QE là đảm bảo rằng các khách hàng QE và CSFLE không cần lưu trữ bất kỳ trạng thái nào, do đó chúng có thể được sử dụng bởi các khách hàng ngắn hạn và nhẹ nhàng (ví dụ, chạy trong các container) và không cần đồng bộ hóa giữa các khách hàng.

Ví dụ : Mục tiêu này nghĩa là QE không lưu trữ bất kỳ trạng thái nào ở phía khách hàng. Điều này có nghĩa là có thể sử dụng QE trên nhiều máy khách khác nhau mà không cần phải đồng bộ hóa trạng thái giữa chúng. Ví dụ, giả sử có một ứng dụng web chạy trên nhiều máy chủ. Mỗi máy chủ này có thể thực hiện các truy vấn đến cơ sở dữ liệu MongoDB mà không cần biết về các truy vấn mà các máy chủ khác đang thực hiện. Điều này giúp tăng khả năng mở rộng của ứng dụng của và giảm bớt sự phức tạp khi xử lý các trạng thái.

* Bảo mật: Mục tiêu cuối cùng của QE là cung cấp bảo mật trong sử dụng với một hồ sơ bảo mật cụ thể. QE nên đáp ứng các mô hình đe dọa giống như CSFLE nhưng với một hồ sơ bảo mật tốt hơn khi có thể.

Ví dụ: Mục tiêu này nghĩa là QE cung cấp bảo mật trong sử dụng với một hồ sơ bảo mật cụ thể. Điều này có nghĩa là ngay cả khi máy chủ MongoDB bị xâm nhập, kẻ tấn công vẫn không thể đọc được dữ liệu nhạy cảm đã được mã hóa. Ngoài ra, QE còn cung cấp bảo mật chống lại các mối đe dọa nội bộ, như nhân viên đội nội gián hoặc những người có quyền truy cập hợp pháp nhưng có ý định xấu. Ví dụ, giả sử một nhân viên của có quyền truy cập vào máy chủ MongoDB. Ngay cả khi họ có thể truy cập vào máy chủ, họ vẫn không thể đọc được dữ liệu nhạy cảm đã được mã hóa nếu họ không có khóa giải mã. Điều này giúp bảo vệ dữ liệu của khỏi các mối đe dọa từ bên trong tổ chức của .

Ví dụ:

Giả sử có một cơ sở dữ liệu MongoDB chứa thông tin về các bệnh nhân trong một bệnh viện. Một số trường dữ liệu, như name (tên), ssn (số an sinh xã hội), và medicalRecords (hồ sơ y tế), là nhạy cảm và muốn mã hóa chúng.

Với QE, có thể mã hóa các trường này ngay tại máy khách trước khi chúng được gửi qua mạng đến máy chủ MongoDB. Điều này có nghĩa là ngay cả khi dữ liệu bị bắt giữ trong quá trình truyền, kẻ tấn công vẫn không thể đọc được dữ liệu nhạy cảm này.

Khi dữ liệu đến máy chủ MongoDB, nó được lưu trữ dưới dạng dữ liệu đã được mã hóa. Điều này có nghĩa là ngay cả khi máy chủ bị xâm nhập, kẻ tấn công vẫn không thể đọc được dữ liệu nhạy cảm này.

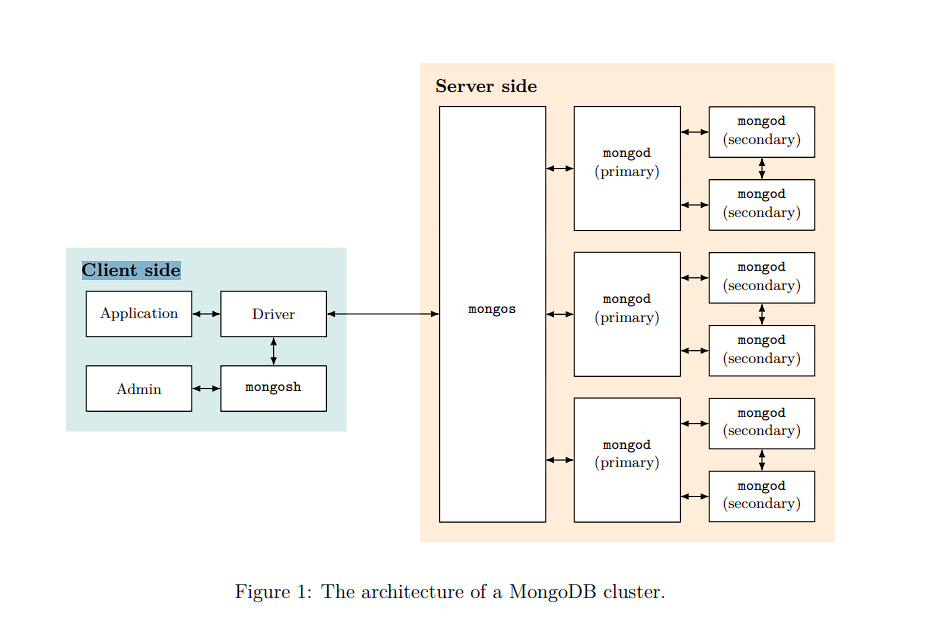
Tuy nhiên, điều đặc biệt về QE là vẫn có thể thực hiện các truy vấn trên các trường đã được mã hóa. Ví dụ, có thể tìm kiếm tất cả các bệnh nhân có tên là “John Doe” mà không cần giải mã dữ liệu. Điều này được thực hiện bằng cách sử dụng các mã thông báo mật mã và các thẻ tìm kiếm mật mã.

Ví dụ, có thể thực hiện truy vấn sau trên cơ sở dữ liệu đã được mã hóa:

db.patients.find({ name: "John Doe" })

Và MongoDB sẽ trả về tất cả các bản ghi có trường name được mã hóa là “John Doe”, mà không cần giải mã dữ liệu.

1. Thiết kế QE



1. **mongod**: Đây là máy chủ cơ sở dữ liệu, nơi lưu trữ dữ liệu và phản hồi các truy vấn. Nhiều phiên bản mongod có thể được kết hợp để tạo thành một bộ sao chép, bao gồm một mongod chính và một số phiên bản mongod phụ lưu trữ các bản sao của dữ liệu. Trong hình, mongod được biểu diễn dưới dạng một nhóm các hộp, với một hộp lớn (mongod chính) và hai hộp nhỏ hơn (mongod phụ). Ví dụ, có thể tưởng tượng mongod như là một thư viện sách, nơi mà mỗi cuốn sách (dữ liệu) được lưu trữ. Thư viện này có thể có nhiều chi nhánh (mongod phụ), và mỗi chi nhánh này đều có các bản sao của cuốn sách.
2. **mongos**: Đây là một thành phần giúp MongoDB mở rộng theo chiều ngang bằng cách chia cơ sở dữ liệu giữa nhiều phiên bản mongod. Trong hình, mongos được biểu diễn dưới dạng một hộp duy nhất ở phía máy chủ. Ví dụ, có thể tưởng tượng mongos như là một người hướng dẫn viên du lịch, người đưa ra chỉ dẫn cho khách du lịch (ứng dụng) để tìm đến địa điểm mong muốn (dữ liệu).
3. **mongosh**: Đây là một shell sử dụng trình điều khiển Node.js và được sử dụng cho các tác vụ quản trị. Trong hình, mongosh được biểu diễn dưới dạng một hộp duy nhất ở phía máy khách. Ví dụ, có thể tưởng tượng mongosh như là một công cụ, giống như một chiếc chìa khóa, giúp quản trị viên truy cập và quản lý cơ sở dữ liệu.
4. **drivers**: Đây là một bộ sưu tập các thư viện cụ thể cho từng ngôn ngữ mà các ứng dụng có thể sử dụng để tương tác với một triển khai MongoDB. Trong hình, drivers được biểu diễn dưới dạng một hộp duy nhất ở phía máy khách. Ví dụ, có thể tưởng tượng drivers như là một bộ dịch, giúp ứng dụng “nói chuyện” với cơ sở dữ liệu bằng ngôn ngữ mà cả hai đều hiểu.
5. **Ứng dụng (Application)**: Đây là chương trình hoặc dịch vụ thực tế mà đang chạy, ví dụ như một trang web hoặc một ứng dụng di động. Ứng dụng này sẽ tương tác với cơ sở dữ liệu MongoDB thông qua driver.
6. **Admin**: Đây là người quản trị hệ thống, người có thể thực hiện các tác vụ quản lý và bảo dưỡng trên cơ sở dữ liệu. Admin tương tác với MongoDB thông qua mongosh.
7. **Driver**: Driver là thư viện mã nguồn mở cung cấp bởi MongoDB, cho phép ứng dụng tương tác với cơ sở dữ liệu MongoDB. Có nhiều loại driver khác nhau tương ứng với các ngôn ngữ lập trình khác nhau như JavaScript, Python, C#, Java, v.v…
8. **mongosh**: Đây là shell của MongoDB, cho phép admin thực hiện các tác vụ quản lý trực tiếp trên cơ sở dữ liệu từ dòng lệnh.

Ví dụ, có thể tưởng tượng ứng dụng là một người đang cố gắng gửi một bức thư (dữ liệu) đi. Driver sẽ là người đưa thư, mang bức thư từ người gửi đến bưu điện (mongos). Mongosh thì giống như một người quản lý bưu điện, người có thể kiểm soát và quản lý cách thức hoạt động của bưu điện.

Các thành phần của QE:

1. **Drivers**: Đây là các thư viện phía máy khách, tương ứng với từng ngôn ngữ, mà ứng dụng sử dụng để tương tác với máy chủ cơ sở dữ liệu. Để sử dụng QE, ứng dụng tải một driver để tạo và tương tác với một máy khách đã mã hóa. Ví dụ, nếu ứng dụng của được viết bằng Python, sẽ sử dụng driver Python để tương tác với MongoDB. Khi muốn sử dụng QE, sẽ tải driver này để tạo và tương tác với một máy khách đã mã hóa.
2. **Máy khách đã mã hóa (Encrypted client)**: Đây là một phiên bản đặc biệt của máy khách MongoDB, được tạo ra bởi driver, có khả năng xử lý dữ liệu đã mã hóa. Máy khách đã mã hóa này chịu trách nhiệm lấy các khóa mã hóa dữ liệu (DEK) đã mã hóa từ một khoá (key vault), và chuyển đổi các thao tác chèn, tìm kiếm, cập nhật và xóa dạng văn bản thuần túy thành các phiên bản đã mã hóa.
3. **Nhà cung cấp KMS (KMS provider)**: Đây là dịch vụ quản lý khóa, như AWS KMS, Azure Key Vault, Google Cloud Platform KMS hoặc KMIP, mà máy khách đã mã hóa sử dụng để quản lý khóa chủ khách hàng (CMK).
4. **Khoá (Key vault)**: Đây là một bộ sưu tập được duy trì tại máy chủ lưu trữ các DEK đã mã hóa với CMK do KMS quản lý. có thể tưởng tượng nó như một két sắt chứa tất cả các khóa mã hóa dữ liệu.
5. **libmongocrypt**: Đây là một thư viện phía máy khách viết bằng C được sử dụng bởi các máy khách đã mã hóa để mã hóa, giải mã và thực hiện các phần phía máy khách của các thao tác chèn, tìm kiếm, cập nhật và xóa đã mã hóa dựa trên lược đồ mã hóa cơ sở dữ liệu OST.
6. **Thư viện phân tích truy vấn chia sẻ (Query analysis shared library)**: Đây là một thư viện phía máy khách cho phép người dùng viết các thao tác chèn, tìm kiếm, cập nhật và xóa dữ liệu đã mã hóa bằng văn bản thuần túy và đánh dấu các phần cần thiết của những truy vấn văn bản thuần túy đó để mã hóa bởi libmongocrypt.

Cách thức hoạt động:

1. **Tạo bộ sưu tập (Create collection)**: Khi tạo một bộ sưu tập đã mã hóa QE, ba bộ sưu tập được tạo: edc lưu trữ các tài liệu đã mã hóa, esc lưu trữ một cấu trúc đã mã hóa phụ trợ cần thiết cho việc tìm kiếm, và ecoc lưu trữ một cấu trúc đã mã hóa phụ trợ cần thiết để nén esc nếu nó quá lớn. có thể tưởng tượng nó như việc tạo ra ba thư mục trên máy tính của mình: một để lưu trữ các tệp tin (edc), một để lưu trữ một danh sách các tệp tin (esc), và một để lưu trữ thông tin về cách tối ưu hóa danh sách đó (ecoc).

edc (Encrypted Data Collection): Đây là bộ sưu tập chính mà sẽ làm việc với nó. Mỗi khi thêm một tài liệu mới vào bộ sưu tập này, các trường nhạy cảm sẽ được mã hóa trước khi được lưu trữ. Ví dụ, nếu đang lưu trữ thông tin người dùng và không muốn mật khẩu của họ có thể được đọc một cách dễ dàng, sẽ lưu trữ mật khẩu đã được mã hóa trong edc.

esc (Encrypted Search Collection): Đây là bộ sưu tập phụ trợ được sử dụng để giúp tìm kiếm các tài liệu đã mã hóa. Mỗi khi thêm một tài liệu mới vào edc, một bản ghi tương ứng sẽ được thêm vào esc. Bản ghi này không chứa dữ liệu thực sự, nhưng thay vào đó là một loại “dấu hiệu” hoặc “chỉ mục” giúp MongoDB tìm kiếm tài liệu đã mã hóa mà không cần phải giải mã chúng.

ecoc (Encrypted Compaction Collection): Đây là bộ sưu tập thứ ba được tạo ra khi kích hoạt QE. Nó chứa thông tin giúp MongoDB nén esc khi cần thiết. Khi esc trở nên quá lớn (điều này có thể xảy ra nếu thêm rất nhiều tài liệu vào edc), MongoDB sẽ sử dụng thông tin trong ecoc để xác định cách tốt nhất để nén esc, giữ cho cơ sở dữ liệu của hoạt động một cách hiệu quả.

1. **Chèn tài liệu (Document insertions)**: Một tài liệu với một tập hợp các trường F với một tập con EF ⊆ F của các trường đã mã hóa QE được chèn như sau. Đối với tất cả các trường f ∈ EF, giá trị vf của f được mã hóa bằng AES256-CBC-SHA256. Ngoài ra, một mảng safeContent lưu trữ các thẻ tìm kiếm mã hóa được thêm vào tài liệu. có thể tưởng tượng nó như việc gửi một bức thư đã được dấu trong một phong bì kín (mã hóa AES256-CBC-SHA256), và cũng gắn một nhãn (thẻ tìm kiếm) lên phong bì để có thể tìm lại nó sau này.
2. **Tìm kiếm (Finds)**: Để tìm các tài liệu phù hợp với một bộ lọc nhất định, một tập hợp các mã thông báo mã hóa được gửi đến máy chủ. Máy chủ sử dụng các mã thông báo này với esc để tạo ra một tập hợp các thẻ tìm kiếm mã hóa và trả về các tài liệu mà mảng safeContent của chúng lưu trữ bất kỳ thẻ nào trong số này. Đây giống như việc đưa cho người khác một danh sách các nhãn (mã thông báo) và họ sẽ tìm trong kho và trả lại cho tất cả các phong bì có nhãn tương ứng.
3. **Cập nhật (Updates)**: Để cập nhật một tài liệu phù hợp với một bộ lọc nhất định, một tập hợp các mã thông báo mã hóa và một giá trị đã mã hóa được cập nhật được gửi đến máy chủ. Máy chủ sử dụng các mã thông báo này với esc để tạo ra một tập hợp các thẻ tìm kiếm mã hóa. Một trong những tài liệu mà safeContent của chúng chứa một trong những thẻ này sau đó được cập nhật với giá trị đã mã hóa mới. Đây giống như việc đưa cho người khác một nhãn và một bức thư mới, và họ sẽ tìm phong bì có nhãn tương ứng và thay thế bức thư cũ bằng bức thư mới.
4. **Xóa (Deletes)**: Để xóa một tài liệu phù hợp với một bộ lọc, một tập hợp các mã thông báo mã hóa và một giá trị đã mã hóa được cập nhật được gửi đến máy chủ. Máy chủ sử dụng các mã thông báo này với esc để tạo ra một tập hợp các thẻ tìm kiếm mã hóa. Một trong những tài liệu mà safeContent của chúng chứa một trong những thẻ này sau đó được xóa. Đây giống như việc đưa cho người khác một nhãn và yêu cầu họ tìm phong bì có nhãn tương ứng và loại bỏ nó.
5. **Nén (Compacts)**: esc tăng kích thước nên QE hỗ trợ một hoạt động nén để giảm kích thước của nó. Để nén esc, một mã thông báo nén mã hóa được gửi đến máy chủ. Máy chủ sử dụng mã thông báo này với thông tin từ ecoc để xóa các mục cũ trong esc. Đây giống như việc đưa cho người khác một danh sách các nhãn không còn cần thiết, và họ sẽ tìm và loại bỏ tất cả các phong bì có nhãn tương ứng.
6. Lý do về bảo mật mã hóa cơ sở dữ liệu

Tại sao mã hóa cơ sở dữ liệu khó thiết kế?

Mã hóa cơ sở dữ liệu, cụ thể là mã hóa cơ sở dữ liệu đang sử dụng, khó thiết kế vì có thể (và thường xảy ra) thiết kế các nguyên tắc/ giao thức mã hóa không an toàn từ các khối xây dựng hoàn toàn an toàn. Điều này có nghĩa là chỉ vì một nguyên tắc/giao thức mã hóa sử dụng một khối xây dựng an toàn không có nghĩa là giao thức đó an toàn. Ví dụ, có thể có một khóa an toàn, nhưng nếu sử dụng nó một cách không chính xác (như để khóa của ở nơi công cộng), thì khóa đó không còn an toàn nữa.

Trong ngữ cảnh của mã hóa cơ sở dữ liệu, điều này dẫn đến sai lầm phổ biến là nghĩ rằng một giải pháp mã hóa cơ sở dữ liệu an toàn chỉ vì nó sử dụng một lược đồ mã hóa tiêu chuẩn như AES. Tuy nhiên, điều quan trọng là hiểu rằng mã hóa AES tiêu chuẩn (ví dụ, sử dụng Chế độ Bộ đếm hoặc CBC) được thiết kế để làm một điều và chỉ một điều: mã hóa một thông điệp với một khóa nhất định để nó có thể được giải mã bằng cùng một khóa. Cụ thể, nó không được thiết kế để hỗ trợ truy vấn trên văn bản mã hóa. Điều này có nghĩa là bất kỳ giải pháp mã hóa cơ sở dữ liệu nào - ngay cả khi hoàn toàn dựa trên AES - phải được xem xét và phân tích như một đối tượng mã hóa mới không tự động kế thừa các thuộc tính an toàn của lược đồ mã hóa cơ bản.

Tại sao mã hóa cơ sở dữ liệu khó phân tích?

Một khi một giải pháp được thiết kế, cần phải phân tích độ an toàn của nó. Độ an toàn của các nguyên tắc mã hóa khó phân tích nói chung và đặc biệt khó đối với mã hóa cơ sở dữ liệu. Điều này là do việc mã hóa cơ sở dữ liệu liên quan đến nhiều yếu tố phức tạp hơn so với mã hóa thông thường. Ví dụ, trong mã hóa thông thường, chỉ cần quan tâm đến việc bảo vệ dữ liệu của mình khỏi những người không được phép truy cập. Nhưng với mã hóa cơ sở dữ liệu, cũng cần phải quan tâm đến việc làm thế nào để thực hiện các truy vấn trên dữ liệu đã mã hóa mà không cần phải giải mã chúng trước. Điều này tạo ra một lớp phức tạp thêm khi phân tích độ an toàn của giải pháp mã hóa cơ sở dữ liệu.

Rò rỉ trong mật mã học

Rò rỉ trong mật mã học: Mỗi lược đồ mã hóa đều rò rỉ một số thông tin về văn bản gốc. Trong trường hợp của AES256-CBC, thông tin rò rỉ là một xấp xỉ về độ dài của thông điệp. Trong trường hợp của one-time pad - một lược đồ mã hóa “hoàn toàn an toàn” - thông tin rò rỉ là độ dài chính xác của thông điệp.

Ví dụ, nếu mã hóa hai thông điệp khác nhau với AES256-CBC, một thông điệp ngắn và một thông điệp dài, kẻ tấn công có thể xác định được thông điệp nào dài hơn chỉ bằng cách nhìn vào độ dài của văn bản mã hóa.

Rò rỉ trong mã hóa cơ sở dữ liệu: Có nhiều cách khác nhau để thiết kế các giải pháp mã hóa cơ sở dữ liệu, nhưng điều quan trọng cần hiểu là tất cả chúng đều rò rỉ một số thông tin. Hơn nữa, rò rỉ không chỉ đặc trưng cho mã hóa cơ sở dữ liệu - nó tồn tại trong mã hóa tiêu chuẩn và các nguyên tắc mật mã và bảo mật khác như tính toán đa phía an toàn và bảo mật phân biệt.

Điều này có nghĩa là, ngay cả khi sử dụng một lược đồ mã hóa tiêu chuẩn như AES để mã hóa cơ sở dữ liệu của mình, vẫn cần phải xem xét những thông tin nào có thể bị rò rỉ và làm thế nào để kiểm soát nó.

Các câu hỏi quan trọng: Câu hỏi không phải là liệu một giải pháp cụ thể có rò rỉ thông tin hay không; thay vào đó, các câu hỏi quan trọng là: (1) nó rò rỉ thông tin gì cụ thể?; (2) liệu thông tin rò rỉ này có thể bị khai thác hay không? và (3) liệu nó có thể được kiểm soát (ưu tiên là một cách hiệu quả) hay không?

Đối với hầu hết các giải pháp, câu trả lời cho những câu hỏi này khó xác định vì nó phụ thuộc vào nhiều yếu tố, bao gồm việc kẻ địch có quyền truy cập vào thông tin gì và thông tin gì mà họ đã biết về dữ liệu.

Đánh đổi:

Hiệu suất: Khi thiết kế một giải pháp mã hóa cơ sở dữ liệu, một yếu tố quan trọng cần xem xét là hiệu suất. Cần phải xác định xem giải pháp này sẽ tạo ra bao nhiêu gánh nặng cho các hoạt động cơ sở dữ liệu và liệu nó có làm tăng kích thước của dữ liệu hay không. Cần lưu ý rằng hiệu suất của một cơ sở dữ liệu đã mã hóa có thể rất khác so với hiệu suất của một cơ sở dữ liệu văn bản gốc, ngay cả khi sử dụng cùng một hệ thống quản lý cơ sở dữ liệu (DBMS). Vì vậy, việc sử dụng giải pháp mã hóa trong sử dụng của DBMS có thể thay đổi một cách đáng kể các công việc mà hệ thống có thể được sử dụng.

Chức năng: Một yếu tố khác cần xem xét là chức năng. Cần xác định xem giải pháp hỗ trợ phần nào của ngôn ngữ truy vấn. Hầu hết các giải pháp mã hóa cơ sở dữ liệu trong sử dụng chỉ hỗ trợ một phần của ngôn ngữ truy vấn của hệ thống và chỉ có thể được sử dụng cho một số ứng dụng nhất định.

Rò rỉ: Một yếu tố quan trọng khác cần xem xét là rò rỉ. Cần xác định xem giải pháp rò rỉ thông tin gì cụ thể và đối với loại kẻ địch nào. Dưới điều kiện nào, thông tin rò rỉ này có thể bị khai thác?

Phương pháp đánh giá:

Phương pháp trực quan (Heuristic): Giải pháp được thiết kế cẩn thận cho một hệ thống quản lý cơ sở dữ liệu thương mại thực tế và để giải quyết một tập hợp các mối đe dọa mà người thiết kế hiểu một cách ngầm định. Giải pháp sau đó được kiểm tra bởi các bên liên quan và được chấp nhận nếu họ cảm thấy thoải mái với thiết kế của nó.

Phương pháp lý thuyết (Theoretical): Giải pháp được thiết kế cẩn thận cho một hệ thống quản lý cơ sở dữ liệu trừu tượng và lý tưởng để giải quyết một tập hợp các mối đe dọa được nắm bắt một cách rõ ràng và chính thức. Giải pháp sau đó được phân tích một cách chính thức liên quan đến mô hình đối thủ bằng cách chính thức hóa hồ sơ rò rỉ của giải pháp, sau đó cung cấp một bằng chứng cho thấy giải pháp rò rỉ tối đa những gì được nắm bắt bởi hồ sơ rò rỉ.

Cả hai phương pháp này đều có ưu và nhược điểm. Phương pháp lý thuyết cung cấp các đảm bảo an ninh rõ ràng, rõ ràng và mạnh mẽ có thể được xác minh, nhưng nó làm điều này trong ngữ cảnh của một hệ thống quản lý cơ sở dữ liệu đơn giản và lý tưởng, không nắm bắt được sự phức tạp của các hệ thống thương mại. Phương pháp trực quan cung cấp các đảm bảo an ninh mơ hồ khó xác minh hơn, nhưng nó làm điều này trong ngữ cảnh của một hệ thống quản lý cơ sở dữ liệu thực sự mà không đưa ra bất kỳ giả định nào về cách hoạt động của hệ thống.

Không ngạc nhiên khi tất cả các công nghệ mã hóa cơ sở dữ liệu thương mại trước đây - bao gồm CSFLE và Always Encrypted for SQL Server - đều được thiết kế và phân tích bằng phương pháp trực quan, và hầu hết các giải pháp mã hóa cơ sở dữ liệu được đề xuất trong văn bản nghiên cứu được thiết kế và phân tích bằng phương pháp lý thuyết. Mặt khác, QE được thiết kế và phân tích bằng một phương pháp lai mới. Mã hóa cơ sở dữ liệu là một lĩnh vực phát triển nhanh chóng và trong khi phương pháp lai của chúng tôi là một bước tiến, nó không hoàn hảo và sẽ tiếp tục phát triển.

Đánh giá hồ sơ rò rỉ thông qua phân tích mật mã

Phương pháp lý thuyết: Phương pháp này chứng minh rằng một giải pháp đạt được một hồ sơ rò rỉ nhất định có giá trị, nhưng chỉ đến một mức độ nhất định. Cụ thể, nó không cho chúng ta biết liệu một hồ sơ rò rỉ cụ thể có thể bị khai thác hay không hoặc thậm chí nó tốt hơn hay tệ hơn hồ sơ khác.

Đánh giá hồ sơ rò rỉ: Việc đánh giá một hồ sơ rò rỉ là một vấn đề khó khăn và chưa có câu trả lời thỏa đáng trong tài liệu nghiên cứu hiện tại. Việc thiết lập một cách chính thức và vững chắc để làm điều này là một trong những vấn đề lớn vẫn còn mở trong lĩnh vực này.

Phân tích mật mã: Hiện tại, phương pháp tốt nhất chúng ta có là sử dụng phân tích mật mã, nói cách khác, là thiết kế các cuộc tấn công rò rỉ cố gắng khai thác các hồ sơ rò rỉ khác nhau. Nếu một hồ sơ rò rỉ chịu được sự kiểm tra, chúng ta có thể có một số sự tự tin rằng nó có thể an toàn. Đây là cách mà hầu hết (nếu không phải tất cả) mật mã thực tế trong thế giới thực hoạt động.

Các ý tưởng nghiên cứu mới: Tuy nhiên, có một số ý tưởng nghiên cứu thú vị về cách đánh giá các hồ sơ rò rỉ theo toán học mà không hoàn toàn dựa vào phân tích mật mã. Những ý tưởng này vẫn còn ở giai đoạn sơ khai nhưng rất triển vọng và có khả năng đóng góp vào sự phát triển của công nghệ mã hóa cơ sở dữ liệu.

Giải thích kết quả phân tích mật mã

Sự xuất hiện của các cuộc tấn công rò rỉ: Các cuộc tấn công rò rỉ là một hình thức mới của phân tích mật mã, vì vậy chúng vẫn chưa được hiểu rõ.

Đánh giá: Không có sự đồng lòng về cách các cuộc tấn công rò rỉ nên được đánh giá theo thực tế. Các bài báo sử dụng các phương pháp khác nhau và thường xuyên, tiêu chí đánh giá được chọn một cách tùy ý và không có lý do.

Giả định: Các cuộc tấn công rò rỉ dựa trên các giả định về dữ liệu và/hoặc truy vấn và những gì kẻ thù đã biết về chúng. Mức độ hợp lý của các giả định này thay đổi và các chuyên gia có thể không đồng ý về việc một giả định cụ thể có hợp lý trong thực tế hay không.

Đoạn văn cũng nhấn mạnh rằng mục tiêu ở đây không phải là giảm nhẹ việc nghiên cứu về các cuộc tấn công rò rỉ. Thực tế, đây là một lĩnh vực nghiên cứu quan trọng cho MongoDB, được minh họa bởi việc các nhà nghiên cứu của MongoDB đã đóng góp vào việc nghiên cứu về các cuộc tấn công rò rỉ.

Cuối cùng, đoạn văn khuyến nghị rằng tác động thực tế của một cuộc tấn công rò rỉ cần được đánh giá một cách cẩn thận. Kết quả mật mã học mà nó đưa ra cần được đặt trong ngữ cảnh bằng cách đặt các câu hỏi sau:

Mô hình kẻ thù mà cuộc tấn công hoạt động trong đó có liên quan đến tình huống triển khai và mô hình đe dọa của tôi không?

Hồ sơ rò rỉ mà cuộc tấn công khai thác có liên quan đến công nghệ mã hóa cơ sở dữ liệu mà tôi đang sử dụng không?

Những hạn chế của cuộc tấn công là gì và chúng có được truyền đạt và tiết lộ đúng cách không?

Cuộc tấn công dựa trên những giả định nào và chúng có thực tế không?

Cuộc tấn công đã được đánh giá thực nghiệm như thế nào và phương pháp đánh giá có ý nghĩa không?

Có biện pháp phòng ngừa đơn giản nào không?

Câu trả lời cho những câu hỏi này đôi khi khó xác định nên luôn cẩn thận làm việc với một chuyên gia trong lĩnh vực để có cái nhìn tổng quan. Điều này không khác gì việc đánh giá và xác định tầm quan trọng thực tế của các cuộc tấn công được công bố đối với các lược đồ mã hóa đối xứng như AES, các hàm băm mật mã học như SHA256 và các lược đồ mã hóa khóa công khai như RSA.

Nắm bắt các mối đe dọa nội bộ (Capturing insider threats)

Mô hình snapshot: Mô hình này nắm bắt việc tiết lộ chính cơ sở dữ liệu đã mã hóa. Tuy nhiên, nó không bao gồm bất kỳ nhật ký hoặc cấu trúc dữ liệu nào do DBMS quản lý. Điều này có nghĩa là nếu có bất kỳ dữ liệu nào được lưu trữ ngoài cơ sở dữ liệu đã mã hóa, nó sẽ không được bảo vệ.

Mô hình persistent: Mô hình này nắm bắt các trường hợp mà máy chủ cơ sở dữ liệu hoàn toàn và hoàn toàn bị xâm phạm. Điều này có nghĩa là nếu một kẻ tấn công có thể kiểm soát hoàn toàn máy chủ cơ sở dữ liệu, họ có thể tiếp cận với tất cả dữ liệu, bao gồm cả dữ liệu đã mã hóa.

Tuy nhiên, những mô hình này không chính xác nắm bắt phần lớn các mối đe dọa mà người dùng cơ sở dữ liệu thương mại quan tâm trong thực tế. Theo Báo cáo Chi phí của Vi phạm Dữ liệu 2022, các mối đe dọa từ bên trong chiếm khoảng 47% số vi phạm và ít nhất 69% số vi phạm không yêu cầu bất kỳ sự xâm phạm nào đối với cơ chế bảo mật cơ bản của hệ thống. Những số liệu thống kê này động viên các mô hình kẻ thù tinh vi hơn và các phân tích bảo mật mà chúng tôi đề xuất và mô tả chi tiết trong Phần 5 và 8.

Điều này có nghĩa là, thay vì chỉ xem xét các mô hình đe dọa đơn giản, chúng ta cần xem xét các mô hình phức tạp hơn để phản ánh đúng hơn thực tế mà người dùng cơ sở dữ liệu thương mại đang phải đối mặt. Điều này bao gồm việc xem xét các mối đe dọa từ bên trong, nơi một người dùng hợp pháp có thể lạm dụng quyền truy cập của họ để lấy dữ liệu, cũng như các trường hợp mà hệ thống bảo mật cơ bản không bị xâm phạm nhưng dữ liệu vẫn bị lộ.

Đánh giá mã hóa có thể truy vấn (Evaluate queryable encryption)

1. Thiết Kế và Phân Tích Lý Thuyết:

Mục Tiêu: Xác định độ an toàn của hệ thống mã hóa cơ sở dữ liệu QE thông qua việc thiết kế và phân tích lý thuyết của giải thuật mật mã cốt lõi OST.

Phương Pháp:

Sử dụng mô hình an ninh chứng minh/giảm bớt để chứng minh rằng OST đạt được một hồ sơ rò rỉ định nghĩa rõ ràng đối với mô hình đối thủ cụ thể, đặc biệt là chống lại đối thủ có khả năng chụp nhiều "snapshot."

2. Đánh Giá Thiết Kế và An Ninh Tự Do:

Mục Tiêu: Đảm bảo rằng thiết kế và phân tích an ninh lý thuyết của OST được xác nhận bởi một chuyên gia mật mã ngoại vi với kinh nghiệm nghiên cứu vững về tìm kiếm được mã hóa.

3. Phân Tích Nội Bộ:

Mục Tiêu: Kiểm tra hệ thống chống lại mối đe dọa từ bên trong.

Phương Pháp:

Xem xét kiến trúc và xác định thông tin liên quan đến cơ sở dữ liệu QE đã được mã hóa trong MongoDB.

Nghiên cứu các vai trò tích hợp sẵn trong MongoDB và các lệnh được ủy quyền cho từng vai trò.

Xác định hồ sơ rò rỉ của từng vai trò dựa trên cấu trúc ủy quyền và kiến trúc ghi chú của MongoDB.

4. Xem xét Mã Nội Bộ:

Mục Tiêu: Đảm bảo rằng mã nguồn thực hiện đúng theo thiết kế.

Phương Pháp:

Sau khi đội kỹ thuật triển khai QE, toàn bộ mã nguồn được xem xét để phát hiện và sửa những khả năng sai lệch giữa triển khai và thiết kế.

5. Kiểm Tra Mã Ngoại Bộ:

Mục Tiêu: Xác nhận tính an toàn và đúng đắn của mã nguồn thông qua một đánh giá bởi một công ty an ninh độc lập.

Ghi Chú:

Mặc dù phương pháp đánh giá đã được mô tả rõ ràng, nhóm đang tiếp tục nỗ lực để phát triển công cụ và phương pháp mới để nâng cao quá trình đánh giá hệ thống.

Cảnh báo rằng phân tích và chứng minh của OST được thực hiện trong môi trường lý tưởng và có giới hạn khi áp dụng vào hệ thống thực tế, nơi mà hệ quản trị cơ sở dữ liệu (DBMS) thực tế có thể ghi log thông tin bổ sung có thể thay đổi hồ sơ rò rỉ được đạt được trong phân tích lý thuyết.

1. Mô hình đối nghịch (Adversarial Models)

Cách phân tích an ninh của một cơ sở dữ liệu được mã hóa thông qua nhiều mô hình đối thủ khác nhau. Các mô hình này có thể được phân loại theo hai chiều:

Quan điểm của đối thủ: Đây là những gì mà đối thủ có thể nhìn thấy, ví dụ như cơ sở dữ liệu, đĩa, một số bản ghi, bộ nhớ hoặc thậm chí toàn bộ máy chủ.

Lịch trình: Đây là thời gian và thời lượng mà đối thủ có quyền truy cập vào quan điểm của mình, ví dụ như một thời điểm cụ thể, nhiều thời điểm khác nhau hoặc liên tục.

Các mô hình đối thủ hiện có trong văn bản gồm có đối thủ kiên trì (persistent adversaries) và đối thủ chụp ảnh nhanh (snapshot adversaries). Đối thủ kiên trì có quyền truy cập vào toàn bộ máy chủ cơ sở dữ liệu, trong khi đối thủ chụp ảnh nhanh chỉ nhận được cơ sở dữ liệu đã được mã hóa.

Tuy nhiên, các mô hình này không nhất thiết phản ánh đúng các hệ thống cơ sở dữ liệu mã hóa trong thực tế. Do đó, chúng ta cần mở rộng và tăng số lượng các mô hình đối thủ cho cơ sở dữ liệu mã hóa để phản ánh tốt hơn các đảm bảo an ninh của hệ thống cơ sở dữ liệu mã hóa thực tế.

Các mô hình đối thủ mới sẽ được mô tả dựa trên hai chiều sau:

Bề mặt (surface): Bề mặt của một quan điểm là tập hợp các thành phần DBMS mà đối thủ có thể xâm nhập. Trong trường hợp xấu nhất, đây có thể là toàn bộ DBMS nhưng trong một số trường hợp khác, nó có thể bao gồm một tập hợp các thành phần như giao diện mạng, cơ sở dữ liệu dạng văn bản, chỉ mục, cơ sở dữ liệu đã mã hóa, các cấu trúc dữ liệu khác nhau, bản ghi, tệp và đĩa.

Lịch trình (schedule): Lịch trình của một quan điểm là tập hợp các thời điểm mà đối thủ có thể truy cập vào các thành phần của bề mặt của mình. Có thể xem xét nhiều lịch trình khác nhau, ví dụ, sau mỗi hoạt động, hoặc chỉ sau một hoạt động cụ thể, hoặc theo một phân phối xác suất nào đó.

Định nghĩa 5.1 giải thích rằng một đối thủ A có một (bề mặt, lịch trình)-quan điểm nếu nó có thể quan sát mọi thành phần trong bề mặt ⊆ dbms tại mỗi thời điểm trong lịch trình. Chúng ta đôi khi nói rằng A là một (bề mặt, lịch trình)-đối thủ.

Sử dụng khung này, chúng ta có thể mô tả các loại đối thủ chụp ảnh cũng như đối thủ kiên trì. Ví dụ, đối thủ chụp ảnh đơn truyền thống là (edb,t)-đối thủ cho t ∈ [n], nơi [n] biểu thị tập hợp các số nguyên {1,… , n}. Đối thủ chụp ảnh nhiều lần truyền thống là (edb,[n])-đối thủ và đối thủ kiên trì là (dbms,o∞)-đối thủ nơi dbms đề cập đến toàn bộ hệ thống (bao gồm cả giao diện mạng) và ∞ là lịch trình liên tục cho phép đối thủ truy cập vào các thành phần bị xâm nhập vào bất kỳ thời điểm nào và trong thời gian dài mà nó muốn.

Giả sử chúng ta có một hệ thống quản lý cơ sở dữ liệu (DBMS) gồm các thành phần sau: Mạng (Network), Cơ sở dữ liệu văn bản (Plaintext Database), Cơ sở dữ liệu đã mã hóa (Encrypted Database), và Bộ nhớ (Memory).

Đối thủ kiên trì (Persistent Adversary): Đối thủ này có thể truy cập vào toàn bộ hệ thống DBMS (mạng, cơ sở dữ liệu văn bản, cơ sở dữ liệu đã mã hóa, và bộ nhớ) và có thể làm điều này liên tục. Điều này có nghĩa là họ có thể quan sát và tương tác với hệ thống bất cứ lúc nào.

Đối thủ chụp ảnh đơn (Single-Snapshot Adversary): Đối thủ này chỉ có thể truy cập vào cơ sở dữ liệu đã mã hóa tại một thời điểm cụ thể. Ví dụ, họ có thể nhận được một bản sao của cơ sở dữ liệu đã mã hóa tại thời điểm t = 5, nhưng không thể truy cập hoặc quan sát bất kỳ thay đổi nào sau thời điểm đó.

Đối thủ chụp ảnh nhiều lần (Multi-Snapshot Adversary): Đối thủ này có thể truy cập vào cơ sở dữ liệu đã mã hóa tại nhiều thời điểm khác nhau. Ví dụ, họ có thể nhận được bản sao của cơ sở dữ liệu đã mã hóa tại các thời điểm t = 1, t = 3, và t = 7, cho phép họ quan sát sự thay đổi của cơ sở dữ liệu qua thời gian.

1. Phân tích lý thuyết (Theoretical Analysis)

Phân tích các giải pháp thế hệ đầu tiên (Analysis of first-generation solutions)

Phần này nói về những hạn chế về bảo mật của các giải pháp mã hóa cơ sở dữ liệu thế hệ đầu tiên như ibmne, TDE và ESE. Những giải pháp này chỉ mã hóa các tệp cơ sở dữ liệu ở cấp độ động cơ lưu trữ bằng một khóa được quản lý bên ngoài. Mục đích của những giải pháp này là để bảo vệ cơ sở dữ liệu khỏi việc bị đánh cắp đĩa hoặc các hình thức xâm nhập khác ở cấp độ đĩa. Tuy nhiên, những giải pháp này cũng tiết lộ một số thông tin về cơ sở dữ liệu cho kẻ tấn công, tùy thuộc vào mô hình kẻ tấn công mà chúng ta xem xét. Ví dụ, TDE và ESE tiết lộ ít nhất là kích thước của cơ sở dữ liệu cho kẻ tấn công có thể truy cập vào đĩa ở một thời điểm nhất định, và tiết lộ nhiều hơn cho kẻ tấn công có thể truy cập vào đĩa ở nhiều thời điểm khác nhau. Điều này xảy ra vì khi có thay đổi nào đó trong cơ sở dữ liệu, TDE/ESE chỉ mã hóa lại những trang bị ảnh hưởng bởi thay đổi đó. Nếu kẻ tấn công có được các bản sao của đĩa trước và sau khi thay đổi, nó có thể biết được số trang bị ảnh hưởng bởi thay đổi bằng cách tìm kiếm những trang thay đổi. Từ đó, số trang thay đổi có thể được liên kết với, ví dụ, số tài liệu/hàng được lọc bởi thao tác cập nhật. Trong thực tế, thông tin rò rỉ này có thể không quan trọng, nhưng nó cho thấy rằng những giải pháp này vẫn tiết lộ thông tin cho kẻ tấn công ngay cả khi nó chỉ có thể truy cập vào đĩa. Điều cần nhớ là mọi giải pháp đều có thông tin rò rỉ, và ngay cả những công nghệ mã hóa cơ sở dữ liệu tiêu chuẩn và được hiểu rõ cũng có những hành vi tinh vi khi được xem xét kỹ lưỡng hơn, tùy thuộc vào mô hình kẻ tấn công mà chúng ta quan tâm.

Phân tích các giải pháp thế hệ thứ 2 (Analysis of second-generation solutions)

Phần này nói về các giải pháp mã hóa cơ sở dữ liệu thế hệ thứ hai như AE, CSFLE và một số hệ thống nghiên cứu trước đây như. Những giải pháp này hỗ trợ truy vấn trên dữ liệu đã được mã hóa bằng cách mã hóa các giá trị bằng mã hóa xác định (deterministic encryption). Mã hóa xác định là một phương pháp mã hóa luôn tạo ra cùng một bản mã cho một bản rõ và khóa nhất định, ngay cả khi thực hiện lại thuật toán mã hóa.

Tuy nhiên, việc sử dụng mã hóa xác định như một giải pháp mã hóa cơ sở dữ liệu chưa bao giờ được phân tích một cách chính thức cho đến nay. Phần này đưa ra một số thông tin rò rỉ tối thiểu của những giải pháp này đối với kẻ tấn công có thể truy cập vào cơ sở dữ liệu đã mã hóa ở nhiều thời điểm khác nhau (edb, [n])-adversaries. Các thông tin rò rỉ này bao gồm:

Các hoạt động khởi tạo (init) không tiết lộ bất cứ điều gì ngoài các tham số công khai.

Các hoạt động chèn (insert) tiết lộ hình dạng của tài liệu được chèn, tức là một cận trên và cận dưới về độ dài của các giá trị của mỗi trường đã mã hóa. Chúng cũng tiết lộ sự bằng nhau về giá trị ở cấp độ trường, tức là liệu hai giá trị đã mã hóa cho cùng một trường trong các tài liệu khác nhau có giống nhau hay không. Đặc biệt, sự bằng nhau về giá trị ở cấp độ trường cũng tiết lộ tần suất của các cặp trường/giá trị trong một bộ sưu tập.

Các hoạt động tìm kiếm (find) không tiết lộ gì cả.

Các hoạt động xóa một (deleteOne) tiết lộ khi nào tài liệu bị xóa được chèn hoặc cập nhật trong quá khứ.

Các hoạt động cập nhật một (updateOne) tiết lộ hình dạng của thao tác cập nhật và khi nào tài liệu được cập nhật được chèn hoặc cập nhật trong quá khứ. Ngoài ra, nó cũng tiết lộ sự bằng nhau về giá trị ở cấp độ trường của các cặp trường/giá trị được cập nhật, từ đó tiết lộ tần suất của chúng.

Lưu ý rằng cả CSFLE và AE hỗ trợ nhiều hơn những hoạt động này, nhưng ở đây, chúng ta chỉ xem xét tập con này vì đây là những gì QE hỗ trợ.

Phân tích mã hóa có thể truy vấn (Analysis of queryable encryption)

Sự khác biệt về bảo mật giữa hai giải pháp mã hóa cơ sở dữ liệu: CSFLE và QE. CSFLE sử dụng mã hóa xác định, trong khi QE sử dụng mã hóa có cấu trúc. Mã hóa xác định là một phương pháp mã hóa luôn tạo ra cùng một bản mã cho một bản rõ và khóa nhất định, ngay cả khi thực hiện lại thuật toán mã hóa1. Mã hóa có cấu trúc là một phương pháp mã hóa một cấu trúc dữ liệu sao cho, với khóa và một truy vấn, người dùng có thể tạo ra một mã truy vấn để truy vấn cấu trúc dữ liệu đã mã hóa.

Đoạn này cho thấy rằng CSFLE tiết lộ mẫu tần suất của dữ liệu cho kẻ tấn công chỉ có thể truy cập vào một bản sao của cơ sở dữ liệu đã mã hóa (edb, t)-adversaries. Điều này có nghĩa là kẻ tấn công có thể biết được số lần xuất hiện của một giá trị nhất định trong cơ sở dữ liệu. Ngược lại, QE không tiết lộ thông tin này cho kẻ tấn công. Thực tế, QE có một hồ sơ rò rỉ tốt hơn đối với cả kẻ tấn công truy cập một lần và nhiều lần. Một phân tích lý thuyết về giải pháp mã hóa có cấu trúc OST của QE có sẵn trong bài báo, nơi nó được chứng minh có hồ sơ rò rỉ sau đây đối với kẻ tấn công truy cập nhiều lần (edb, [n])-adversaries:

Các hoạt động khởi tạo (init) không tiết lộ bất cứ điều gì ngoài các tham số công khai.

Các hoạt động chèn (insert) tiết lộ hình dạng của tài liệu được chèn.

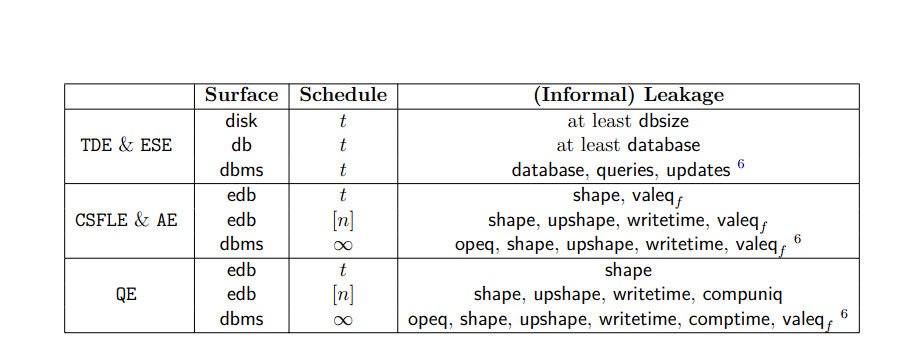
Các hoạt động tìm kiếm (find) không tiết lộ gì cả.

Các hoạt động xóa một (deleteOne) tiết lộ khi nào tài liệu bị xóa được chèn hoặc cập nhật trong quá khứ.

Các hoạt động cập nhật một (updateOne) tiết lộ hình dạng của thao tác cập nhật và khi nào tài liệu được cập nhật được chèn hoặc cập nhật trong quá khứ.

Các hoạt động nén (compact) tiết lộ số lượng các giá trị duy nhất cho tất cả các trường được chèn kể từ lần nén cuối cùng.

So sánh với các giải pháp thế hệ hai:



bảng so sánh các hồ sơ rò rỉ của các công nghệ mã hóa cơ sở dữ liệu thế hệ đầu tiên, thứ hai và thứ ba đối với nhiều loại kẻ tấn công khác nhau. Bảng này có ba cột: Bề mặt (Surface), Lịch trình (Schedule) và Rò rỉ (Informal Leakage). Bề mặt là tập hợp các thành phần DBMS mà kẻ tấn công có thể xâm nhập, lịch trình là tập hợp các thời điểm mà kẻ tấn công có thể truy cập vào các thành phần của bề mặt, và rò rỉ là những thông tin mà kẻ tấn công có thể biết được từ việc truy cập vào bề mặt.

Bảng này liệt kê các giải pháp mã hóa cơ sở dữ liệu như TDE, ESE, CSFLE, AE và QE, và cho biết bề mặt, lịch trình và rò rỉ của chúng. Các mẫu rò rỉ bao gồm:

* dbsize: tiết lộ kích thước của cơ sở dữ liệu.
* database: tiết lộ toàn bộ cơ sở dữ liệu.
* shape: tiết lộ cận trên và cận dưới về độ dài của các giá trị.
* valeqf: tiết lộ liệu hai cặp trường/giá trị có bằng nhau hay không.
* upshape: tiết lộ cận trên và cận dưới về các giá trị được cập nhật của một tài liệu được cập nhật.
* writetime: tiết lộ khi nào một tài liệu được chèn, cập nhật hoặc xóa.
* opeq: tiết lộ nếu và khi nào hai hoạt động được thực hiện trên cùng một tham số (ví dụ, cặp trường/giá trị, bộ lọc, v.v.).
* compuniq: tiết lộ số lượng các giá trị duy nhất được chèn kể từ lần nén cuối cùng.
* comptime: tiết lộ khi nào một hoạt động nén xảy ra.

Bảng này chỉ mang tính minh họa ở mức độ cao và không hoàn toàn chính xác và chặt chẽ.

Giả sử có một cơ sở dữ liệu MongoDB chứa các tài liệu về các sản phẩm của một cửa hàng trực tuyến. Mỗi tài liệu có các trường như tên, giá, mô tả, hình ảnh, số lượng, v.v. muốn bảo vệ cơ sở dữ liệu này khỏi việc bị đánh cắp hoặc xâm nhập bằng cách sử dụng một trong ba phương pháp mã hóa sau:

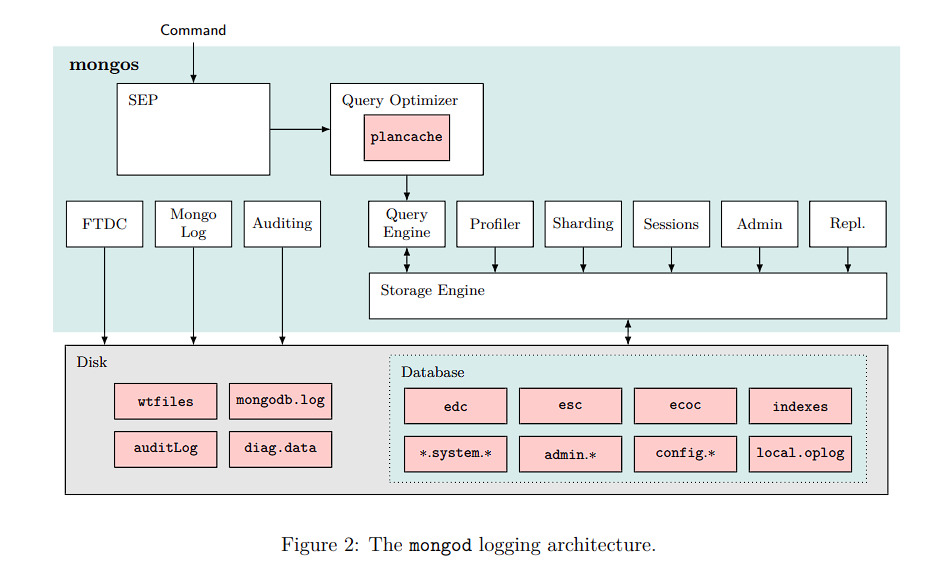
TDE: sẽ mã hóa toàn bộ tệp cơ sở dữ liệu bằng một khóa được quản lý bên ngoài. Khi truy vấn cơ sở dữ liệu, sẽ cần khóa để giải mã tệp và lấy dữ liệu. Tuy nhiên, nếu có một kẻ tấn công nào đó có thể truy cập vào đĩa của , anh ta sẽ có thể biết được kích thước của cơ sở dữ liệu, và nếu anh ta có thể truy cập vào hệ thống DBMS của , anh ta sẽ có thể biết được toàn bộ nội dung, truy vấn và cập nhật của cơ sở dữ liệu.

ESE: sẽ mã hóa các trang cơ sở dữ liệu bằng một khóa được quản lý bên ngoài. Khi truy vấn cơ sở dữ liệu, sẽ cần khóa để giải mã các trang và lấy dữ liệu. Tuy nhiên, nếu có một kẻ tấn công nào đó có thể truy cập vào đĩa của , anh ta sẽ có thể biết được kích thước của cơ sở dữ liệu, và nếu anh ta có thể truy cập vào hệ thống DBMS, anh ta sẽ có thể biết được toàn bộ nội dung, truy vấn và cập nhật của cơ sở dữ liệu.

QE:sẽ mã hóa các giá trị của các trường trong cơ sở dữ liệu bằng một khóa được quản lý bên ngoài. Khi truy vấn cơ sở dữ liệu, sẽ cần khóa để giải mã các giá trị và lấy dữ liệu. Ngoài ra, sẽ cần tạo ra các bộ sưu tập phụ trợ đã mã hóa để hỗ trợ truy vấn trên dữ liệu đã mã hóa. Nếu có một kẻ tấn công nào đó có thể truy cập vào cơ sở dữ liệu đã mã hóa của , anh ta sẽ chỉ có thể biết được hình dạng, thời gian ghi, sự duy nhất sau nén và thời gian nén của cơ sở dữ liệu, nhưng không thể biết được nội dung cụ thể của các giá trị.

1. Kiến trúc ghi nhật ký của MongoDB ( the MongoDB Logging Architecture)

Kiến trúc mongod



* **Command** là hộp màu xanh lá cây ở trên cùng bên trái của hình, có nhãn là “mongos”. Đây là thành phần nhận các lệnh từ người dùng hoặc ứng dụng và chuyển chúng đến **SEP** và **Query Optimizer**.
* **SEP** (Storage Engine Proxy) là hộp màu xanh lá cây ở trên cùng bên phải của hình, có nhãn là “SEP”. Đây là thành phần xử lý các lệnh và kiểm tra xem người dùng có được phép thực hiện các lệnh đó hay không. Nếu lệnh là một hoạt động CRUD (Create, Read, Update, Delete) và người dùng được phép, lệnh sẽ được chuyển đến **Query Optimizer**.
* **Query Optimizer** là hộp màu xanh lá cây ở giữa bên trái của hình, có nhãn là “Query Optimizer”. Đây là thành phần tìm kiếm một kế hoạch truy vấn tối ưu để thực hiện truy vấn một cách hiệu quả. Trong quá trình đó, **Query Optimizer** có thể lưu trữ kế hoạch truy vấn mà nó tìm thấy trong **plancache**, là hộp màu xanh lá cây nhỏ ở dưới **Query Optimizer**. Sau khi tìm được kế hoạch truy vấn, nó sẽ chuyển nó đến **Query Engine** để thực thi.
* **Query Engine** là hộp màu hồng ở giữa bên phải của hình, có nhãn là “Query Engine”. Đây là thành phần thực thi kế hoạch truy vấn cùng với **Storage Engine**. **Query Engine** cũng ghi lại các hoạt động truy vấn vào **Mongo Log**, **Database Profiler** và **Auditing**, là các hộp màu hồng nhỏ ở trên **Query Engine**.
* **Storage Engine** là hộp màu hồng lớn ở dưới cùng của hình, có nhãn là “Storage Engine”. Đây là thành phần quản lý việc lưu trữ và truy xuất dữ liệu trên đĩa và bộ nhớ. Nó bao gồm nhiều thành phần khác nhau như **FTDC**, **Mongo Log**, **Auditing**, **Profiler**, **Sharding**, **Sessions**, **Admin** và **Repl**. Các thành phần này có các chức năng khác nhau như thu thập dữ liệu chẩn đoán, ghi lại các sự kiện hệ thống, hỗ trợ sao chép và phân mảnh dữ liệu, quản lý người dùng và vai trò, v.v.
* **Database** là hộp màu xanh lá cây lớn ở dưới cùng bên phải của hình, có nhãn là “Database”. Đây là thành phần chứa các tệp và thành phần cơ sở dữ liệu như **wtfiles** (WiredTiger files), **mongodb.log**, **auditLog**, **diag.data**, **edc** (Encryption Data Cache), **esc** (Encryption Storage Cache), **eocc** (Encryption Operation Count Cache), **indexes**, ***.system.***, **admin.***,****config:***, **local.oplog**.
* **auditing service** : Đây là một dịch vụ ghi lại các hoạt động của hệ thống như các sự kiện xác thực, tạo, đổi tên và xóa bộ sưu tập, tạo chỉ số, tạo vai trò, cấp và thu hồi quyền, và khởi động và tắt hệ thống. Dịch vụ kiểm toán cũng có thể ghi lại các hoạt động cơ sở dữ liệu như chèn, tìm, tìm và sửa, v.v. Dịch vụ kiểm toán được tắt theo mặc định và phải được bật một cách rõ ràng. Nhật ký kiểm toán có thể được xuất ra màn hình, syslog hoặc một tệp, ví dụ như data/db/auditLog.json.
* **Replication** là quá trình sao chép dữ liệu từ một mongod instance sang các mongod instance khác để đảm bảo tính dự phòng và khả dụng của cơ sở dữ liệu. MongoDB hỗ trợ replication bằng cách sử dụng replica set, là một nhóm các mongod instance có chứa cùng một dữ liệu. Trong một replica set, có một instance chính (primary) và các instance phụ (secondaries). Instance chính là nguồn duy nhất cho các hoạt động ghi, còn các instance phụ sẽ sao chép dữ liệu từ instance chính và có thể phục vụ các hoạt động đọc. Nếu instance chính bị lỗi, một trong các instance phụ sẽ được bầu làm instance chính mới
* **Config** là một cơ sở dữ liệu đặc biệt mà MongoDB sử dụng để lưu trữ các thông tin cấu hình cho việc phân mảnh (sharding) và phiên nhất quán nguyên nhân (causally consistent sessions). Phân mảnh là quá trình chia nhỏ dữ liệu thành các phần nhỏ hơn được gọi là shard, để phân tán dữ liệu trên nhiều máy chủ và tăng khả năng mở rộng của cơ sở dữ liệu. Phiên nhất quán nguyên nhân là một tính năng cho phép các hoạt động đọc và ghi trên cùng một phiên được thực hiện theo một thứ tự nhất định2. Cơ sở dữ liệu config bao gồm nhiều bộ sưu tập khác nhau để lưu trữ các thông tin về các shard, các chunk, các zone, các phiên, v.v.
* **FTDC (Full Time Diagnostic Data Capture)** là một dịch vụ mà mongod chạy để thu thập các số liệu về sử dụng CPU, bộ nhớ và đĩa, hiệu năng mạng và hoạt động CRUD. FTDC giúp hỗ trợ việc giải quyết các vấn đề về cơ sở dữ liệu bằng cách cung cấp các dữ liệu chẩn đoán. FTDC lưu trữ các số liệu trong một thư mục gọi là diagnostic.data trên hệ thống tệp. Theo mặc định, FTDC thu thập thông tin mỗi giây và thư mục diagnostic.data có giới hạn 200MB trước khi bị ghi đè. FTDC có thể bị tắt trên cả mongos và mongod
* **Admin** là một cơ sở dữ liệu đặc biệt mà MongoDB sử dụng để lưu trữ các dữ liệu xác thực và ủy quyền người dùng, như tên người dùng và mật khẩu. Nó cũng chứa các thông tin về các vai trò và đặc quyền được gán cho mỗi người dùng.
* **Other database-specific system collections** là các bộ sưu tập hệ thống nội bộ mà MongoDB duy trì cho mỗi cơ sở dữ liệu cụ thể. Một số ví dụ về các bộ sưu tập này là:
  + \***.system.js**: lưu trữ mã JavaScript do người dùng định nghĩa cho việc tính toán ở phía máy chủ.
  + \***.system.views**: chứa các thông tin về các view của cơ sở dữ liệu. View là một loại bộ sưu tập ảo được tạo ra từ các truy vấn trên các bộ sưu tập khác.
* **WiredTiger** là động cơ lưu trữ mặc định của MongoDB. WiredTiger lưu trữ tất cả các bộ sưu tập được mô tả ở trên trên đĩa và cũng tạo ra các tệp bổ sung bao gồm journal, diagnostic.data, WiredTigerHS.wt, WiredTigerLAS.wt và các tệp khác cho mục đích cấu hình.

Ví dụ:

Bước 1: Khi gửi một lệnh, ví dụ như db.collection.insertOne({name: "Alice"}), đến một mongod instance, lệnh sẽ được xử lý bởi SEP. SEP sẽ kiểm tra xem có quyền thực hiện lệnh này hay không, và nếu có, nó sẽ chuyển lệnh đến Query Optimizer. SEP cũng sẽ ghi nhật kí lệnh này vào command log, là một tệp nhật kí lúc chứa các lệnh được gửi đến mongod.

Bước 2: Query Optimizer sẽ tìm kiếm một kế hoạch truy vấn tối ưu để thực hiện lệnh của . Nó sẽ sử dụng các chỉ số và các kế hoạch truy vấn đã được lưu trữ trong plancache để giúp việc tìm kiếm nhanh hơn. Sau khi tìm được kế hoạch truy vấn, nó sẽ chuyển kế hoạch đến Query Engine để thực thi. Query Optimizer cũng sẽ ghi nhật kí các thông tin về kế hoạch truy vấn vào query log, là một tệp nhật kí lúc chứa các kế hoạch truy vấn được tạo ra bởi Query Optimizer.

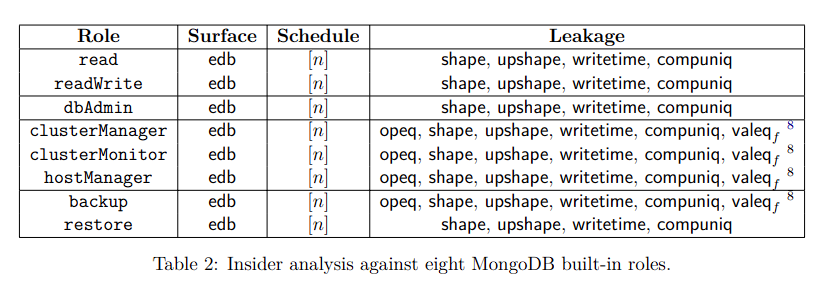
Bước 3: Query Engine sẽ thực thi kế hoạch truy vấn cùng với Storage Engine. Storage Engine sẽ quản lý cách dữ liệu được lưu trữ trên đĩa và trong bộ nhớ. MongoDB sử dụng WiredTiger làm công cụ lưu trữ mặc định. WiredTiger sẽ ghi nhật kí các thay đổi dữ liệu vào các WiredTiger log files, là các tệp nhật kí lúc chứa các thay đổi dữ liệu được thực hiện bởi Storage Engine. WiredTiger cũng sẽ ghi nhật kí các lỗi vào error log, là một tệp nhật kí lỗi chứa các thông báo lỗi của mongod. Query Engine cũng sẽ ghi nhật kí các hoạt động truy vấn vào Mongo Log, Database Profiler và Auditing, là các tệp nhật kí lúc chứa các thông tin về các hoạt động truy vấn, cấu hình, quản trị và kiểm toán của mongod.

Commands: một số lệnh mà có thể chạy để quản lý các cơ sở dữ liệu và cụm MongoDB. Đây không phải là danh sách đầy đủ và chỉ tập trung vào các lệnh liên quan đến phân tích bên trong

* **$collStats**: trả về các số liệu thống kê về một bộ sưu tập nhất định, như kích thước, số lượng tài liệu, số lượng chỉ số, v.v.
* **$planCacheStats**: trả về các số liệu thống kê về kế hoạch truy vấn, như số lượng kế hoạch được lưu trữ, số lần truy cập, số lần thay đổi, v.v.
* **trafficRecord**: ghi lại các giao tiếp mạng (đã được giải mã) giữa các mongod và mongos instance.
* **auditConfigure**: cấu hình kiểm toán cho các mongod và mongos instance trong thời gian chạy. Kiểm toán là một tính năng cho phép ghi lại các hoạt động của hệ thống, như xác thực, tạo và xóa bộ sưu tập, tạo và cấp quyền vai trò, khởi động và tắt hệ thống, v.v.
* **listIndexes**: trả về các thông tin về các chỉ số của một bộ sưu tập nhất định, bao gồm các khóa và tùy chọn được sử dụng để tạo chỉ số.
* **$renameCollectionSameDB**: cho phép người dùng thay đổi tên của một bộ sưu tập hiện có trên cơ sở dữ liệu hiện tại.
* **serverStatus**: trả về các số liệu thống kê được thu thập bởi dịch vụ FTDC, bao gồm các thông tin liên quan đến sử dụng CPU, đĩa và bộ nhớ. Điều này bao gồm độ trễ đọc và ghi, số liệu thống kê về bầu cử nút chính, thông tin khóa và số bit tràn ra đĩa. Lưu ý rằng dữ liệu này được tích lũy trên một máy đơn và không liên quan đến một cơ sở dữ liệu hoặc bộ sưu tập cụ thể. Tuy nhiên, serverStatus có thể được thực thi nhiều lần để thu thập các số liệu thống kê tinh vi hơn. Các số liệu thống kê này vẫn sẽ trên một máy đơn nhưng có thể ở mức độ chi tiết của một giây.
* **setParameter**: thiết lập một loạt các tham số máy chủ. Để xem danh sách đầy đủ, hãy xem [tài liệu](https://www.mongodb.com/developer/products/mongodb/cheat-sheet/).
* **$allCollectionStats**: trả về các số liệu thống kê trên tất cả các bộ sưu tập.
* **$operationMetrics**: trả về các số liệu thống kê nội bộ về thanh toán cho Atlas serverless, như số lượng tài liệu được đọc, số lượng mục chỉ số được đọc hoặc số lần tìm con trỏ.
* **$currentOp**: trả về các thông tin về các hoạt động đang tiến hành.
* **top**: trả về các số liệu thống kê sử dụng cho mỗi bộ sưu tập.
* **getLog**: trả về 1024 sự kiện mongod được ghi nhật kí gần đây nhất từ một cấu trúc dữ liệu được lưu trữ.
* **$connPoolStats**: trả về các thông tin về các kết nối mở đến cụm và bộ sao chép.
* **setClusterParameter**: cho phép người dùng thay đổi tham số changeStreamOptions của một cụm. Tham số này thay đổi chính sách lưu giữ của hình ảnh tài liệu trước và sau khi thay đổi trong một luồng thay đổi.

1. Phân tích nội bộ (Insider Analysis)

Insider analysis against eight MongoDB built-in roles.



Bảng này cho thấy kết quả của phân tích bên trong đối với tám vai trò được xây dựng sẵn trong MongoDB. Phân tích bên trong là một kỹ thuật để đánh giá mức độ rò rỉ thông tin của một hệ thống khi bị tấn công bởi một kẻ xâm nhập có quyền truy cập hợp pháp. Các giá trị trong bảng là:

Vai trò (role): là tên của vai trò được xây dựng sẵn trong MongoDB. Mỗi vai trò có một tập hợp các tài nguyên và lệnh được ủy quyền để truy cập và quản lý cơ sở dữ liệu. Ví dụ, vai trò read chỉ có quyền đọc dữ liệu từ các bộ sưu tập, còn vai trò readWrite có quyền đọc và ghi dữ liệu.

Bề mặt (surface): là tập hợp các tài nguyên và đầu ra của các lệnh mà vai trò có thể truy cập. Bề mặt được biểu diễn bằng một ký hiệu, ví dụ edb là tất cả các bộ sưu tập được mã hóa trong cơ sở dữ liệu, opeq là tất cả các hoạt động ghi trên các trường được mã hóa.

Lịch trình (Schedule): là tần suất mà vai trò thực hiện các lệnh trên bề mặt. Lịch trình được biểu diễn bằng một số nguyên, ví dụ [n] là n lần mỗi ngày.

Rò rỉ (leakage): là tập hợp các thông tin bị tiết lộ cho kẻ xâm nhập khi thực hiện các lệnh trên bề mặt. Rò rỉ được biểu diễn bằng một tập hợp các ký hiệu, ví dụ shape là hình dạng của bộ sưu tập, upshape là hình dạng của thao tác cập nhật, writetime là thời gian ghi, compuniq là số lượng giá trị duy nhất của một trường.

1. Vai trò của người dùng trong cơ sở dữ liệu

2 vai trò người dùng trong cơ sở dữ liệu: vai trò “read” (đọc) và vai trò “readWrite” (đọc và ghi).

1. **Vai trò Read**: Người dùng có vai trò này có thể thực hiện các thao tác đọc trên một tập hợp các cơ sở dữ liệu được ủy quyền và các bộ sưu tập quản lý liên quan của chúng.
   * **Bề mặt (Surface)**: Bao gồm các cơ sở dữ liệu được ủy quyền và các bộ sưu tập *.system.js liên quan. Khi tạo một vai trò đọc, có thể ủy quyền cho nó truy cập vào các cơ sở dữ liệu config.*, local.\* và admin.*. Tuy nhiên, giả định rằng vai trò đọc không được phép truy cập vào cơ sở dữ liệu local.*. Bề mặt cũng bao gồm đầu ra của các lệnh được ủy quyền như $collStats và $planCacheStats.
   * **Lịch trình (Schedule)**: Giả định rằng vai trò đọc có thể truy cập vào bề mặt của mình sau mỗi thao tác.
   * **Chế độ xem (View)**: Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của vai trò đọc chứa dữ liệu liên quan đến QE là edb = (edc, esc, ecoc). Hơn nữa, không có thao tác nào được ủy quyền của vai trò đọc trả về dữ liệu liên quan đến QE. Do đó, chế độ xem của vai trò đọc là của một kẻ thù (edb, [n]).
2. **Vai trò ReadWrite**: Người dùng có vai trò này có thể thực hiện các thao tác đọc và ghi trên một tập hợp các cơ sở dữ liệu được ủy quyền và các bộ sưu tập quản lý liên quan của chúng.
   * **Bề mặt (Surface)**: Tương tự như vai trò đọc, nhưng khi tạo một vai trò readWrite, có thể cấp thêm quyền truy cập vào các cơ sở dữ liệu config.*, local.* và admin.*. Tuy nhiên, giả định rằng vai trò readWrite không được phép truy cập vào cơ sở dữ liệu local.*.
   * **Lịch trình (Schedule)**: Giống như vai trò đọc, giả định rằng vai trò readWrite có thể truy cập vào bề mặt của mình sau mỗi thao tác.
   * **Chế độ xem (View)**: Tương tự như vai trò đọc, chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của vai trò readWrite chứa dữ liệu liên quan đến QE là edb = (edc, esc, ecoc). Do đó, chế độ xem của vai trò readWrite cũng là của một kẻ thù (edb, [n]).

**Ví dụ**: Vai trò Read: Giả sử chúng ta có một người dùng tên là “UserA” với vai trò “read” trên cơ sở dữ liệu “Database1”. UserA có thể thực hiện các thao tác đọc trên Database1, nhưng không thể thực hiện các thao tác ghi. Ví dụ, UserA có thể xem danh sách các bản ghi trong một bảng, nhưng không thể thêm, chỉnh sửa hoặc xóa bản ghi.

Vai trò ReadWrite: Giả sử chúng ta có một người dùng khác tên là “UserB” với vai trò “readWrite” trên cùng một cơ sở dữ liệu “Database1”. Khác với UserA, UserB không chỉ có thể đọc dữ liệu từ Database1, mà còn có thể thực hiện các thao tác ghi. Ví dụ, UserB có thể thêm một bản ghi mới vào một bảng, chỉnh sửa một bản ghi hiện có, hoặc xóa một bản ghi.

1. Vai trò của admin (Database Administration Roles)

Vai trò “dbAdmin” trong cơ sở dữ liệu. Người dùng có vai trò này có thể thực hiện các thao tác quản trị trên một tập hợp các cơ sở dữ liệu được ủy quyền và các bộ sưu tập quản lý liên quan của chúng.

* **Bề mặt (Surface)**: Bao gồm các cơ sở dữ liệu được ủy quyền và bộ sưu tập \*.system.profile liên quan nếu trình phân tích đang được bật. Bề mặt cũng bao gồm đầu ra của các lệnh được ủy quyền như $collStats, $planCacheStats, listIndexes và renameCollectionSameDB.
* **Lịch trình (Schedule)**: Giả định rằng dbAdmin có thể truy cập vào các phần tử của bề mặt của mình sau mỗi thao tác.
* **Chế độ xem (View)**: Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của dbAdmin chứa dữ liệu liên quan đến QE là edb = (edc, esc, ecoc). Hơn nữa, các thao tác được ủy quyền chỉ trả về thông tin (ví dụ, thống kê) về edb. Do đó, chế độ xem của dbAdmin là của một kẻ thù (edb, [n]).

**Ví dụ** : Vai trò dbAdmin: giả sử chúng ta có một người dùng tên là “UserC” với vai trò “dbAdmin” trên Database1. UserC có quyền thực hiện các thao tác quản trị trên Database1. Ví dụ, UserC có thể thay đổi cấu trúc của bảng, tạo hoặc xóa chỉ mục, hoặc thực hiện các thao tác quản lý khác như sao lưu và phục hồi dữ liệu.

1.Vai trò của admin cluster (Cluster Administration Roles)

Vai trò quản lý cụm (Cluster manager role):

vai trò “clusterManager” trong quản lý cụm MongoDB. Người dùng có vai trò này có thể quản lý một cụm.

* **Bề mặt (Surface)**: Bao gồm tất cả các cơ sở dữ liệu trên một cụm, bao gồm cơ sở dữ liệu người dùng và các bộ sưu tập quản lý liên quan như *.system.profile và .system.js. Nó cũng bao gồm các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin.* và local.\*. Mức độ truy cập chính xác là khác nhau nhưng, cho mục đích của chúng ta, chúng ta giả định nó có quyền truy cập đầy đủ vào các cơ sở dữ liệu này để nắm bắt trường hợp tồi tệ nhất. clusterManager có thể chạy các lệnh quản trị như addShard, listShards, removeShard, $planCacheStats và setClusterParameter.
* **Lịch trình (Schedule)**: Chúng ta giả định rằng clusterManager có thể truy cập vào bề mặt của mình sau mỗi thao tác.
* **Chế độ xem (View)**: Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của clusterManager bao gồm dữ liệu liên quan đến QE là edb = (edc, esc, ecoc) và local.oplog. Lưu ý rằng local.oplog lưu trữ số giao dịch, đây là các định danh duy nhất trên toàn phiên được tạo ra bằng một bộ đếm. Số giao dịch được tạo và gán ngay cả cho các giao dịch thất bại nên sự khác biệt giữa hai số giao dịch là một hàm của số lượng giao dịch thất bại, điều này tương quan với số lượng chèn/cập nhật đồng thời trên một cặp trường/giá trị, điều này, lần lượt, tương quan với tần suất của cặp trường/giá trị đó. Những tương quan này không chặt chẽ và bị ảnh hưởng bởi nhiều biến số nhưng, trong trường hợp tồi tệ nhất, có thể tiết lộ thông tin về tần suất. Không có thao tác nào được ủy quyền trả về bất kỳ dữ liệu liên quan đến QE nào. Do đó, chế độ xem của nó là của một kẻ thù (dbms,∞).

Ghi chú: Phân tích của chúng ta về clusterManager giả định rằng nó có quyền truy cập đầy đủ vào local.*, điều này không phải là trường hợp trong thực tế. Cụ thể hơn, clusterManager chỉ có thể cập nhật local.*, nhưng một người có thể sử dụng cập nhật để tìm hiểu thông tin một phần bằng cách phát hành cập nhật cho các cặp trường/giá trị đã biết và quan sát xem cập nhật có được thực hiện hay không. Vì vậy, chúng ta đã mô hình hóa trường hợp tồi tệ nhất và giả định quyền truy cập đầy đủ, nhưng, trong thực tế, điều này rất “bi quan”. Trong công việc tiếp theo, chúng tôi sẽ cung cấp một phân tích chính xác hơn về vai trò này.

**Ví dụ**:

Giả sử chúng ta có một cụm MongoDB với ba cơ sở dữ liệu: “Database1”, “Database2” và “Database3”. Chúng ta có một người dùng tên là “UserD” với vai trò “clusterManager”.

* Bề mặt (Surface): UserD có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm, bao gồm “Database1”, “Database2”, “Database3” và các bộ sưu tập quản lý liên quan như *.system.profile và .system.js. Nó cũng bao gồm các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin.* và local.\*. Ví dụ, UserD có thể xem, thêm, chỉnh sửa hoặc xóa bản ghi trong bất kỳ bảng nào trên bất kỳ cơ sở dữ liệu nào trên cụm.
* Lịch trình (Schedule): Chúng ta giả định rằng UserD có thể truy cập vào bề mặt của mình sau mỗi thao tác. Ví dụ, sau khi thêm một bản ghi vào “Database1”, UserD có thể xem bản ghi mới đó.
* Chế độ xem (View): Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của UserD chứa dữ liệu liên quan đến QE là edb = (edc, esc, ecoc) và local.oplog. Ví dụ, nếu local.oplog lưu trữ số giao dịch, UserD có thể xem số giao dịch này.

2.Vai trò giám sát cụm ( cluster monitor roles)

Vai trò “clusterMonitor” trong quản lý cụm MongoDB. Người dùng có vai trò này có thể giám sát một cụm.

* **Bề mặt (Surface)**: UserD có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm, bao gồm cơ sở dữ liệu người dùng và các bộ sưu tập quản lý liên quan như *.system.profile và .system.js. Nó cũng bao gồm các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin.* và local.\*. clusterMonitor có thể chạy một số lượng lớn hơn và tập hợp các lệnh mạnh mẽ hơn so với clusterManager, bao gồm serverStatus, $collStats, $allCollectionStats, $operationMetrics, $currentOp, top, getLog, và $connPoolStats.
* **Lịch trình (Schedule)**: Chúng ta giả định rằng clusterMonitor có thể truy cập vào bề mặt của mình sau mỗi thao tác.
* **Chế độ xem (View)**: Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của clusterMonitor bao gồm dữ liệu liên quan đến QE là edb = (edc, esc, ecoc) và local.\* bao gồm local.oplog. Các đầu ra của serverStatus, $operationMetrics và $connPoolStats chứa dữ liệu liên quan đến QE. Do đó, chúng ta giả định trường hợp tồi tệ nhất và bao gồm các bản gốc của tất cả các thao tác truy vấn trong chế độ xem. Tóm lại, trong trường hợp tồi tệ nhất, chế độ xem của clusterMonitor là chế độ xem của một kẻ thù (dbms,∞).

**Ví dụ**:

Giả sử chúng ta có một cụm MongoDB với ba cơ sở dữ liệu: “Database1”, “Database2” và “Database3”. Chúng ta có một người dùng tên là “UserE” với vai trò “clusterMonitor”.

Bề mặt (Surface): UserE có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm, bao gồm “Database1”, “Database2”, “Database3” và các bộ sưu tập quản lý liên quan như .system.profile và .system.js. Nó cũng bao gồm các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin. và local.\*. UserE có thể chạy một số lượng lớn hơn và tập hợp các lệnh mạnh mẽ hơn so với clusterManager, bao gồm serverStatus, $collStats, $allCollectionStats, $operationMetrics, $currentOp, top, getLog, và $connPoolStats.

Lịch trình (Schedule): Chúng ta giả định rằng UserE có thể truy cập vào bề mặt của mình sau mỗi thao tác. Ví dụ, sau khi thêm một bản ghi vào “Database1”, UserE có thể xem bản ghi mới đó.

Chế độ xem (View): Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của UserE chứa dữ liệu liên quan đến QE là edb = (edc, esc, ecoc) và local.\* bao gồm local.oplog. Ví dụ, nếu local.oplog lưu trữ số giao dịch, UserE có thể xem số giao dịch này.

4.Vai trò quản lý máy chủ (host manager roles)

Vai trò “hostManager” trong quản lý cụm MongoDB. Người dùng có vai trò này có thể giám sát và quản lý cụm.

* **Bề mặt (Surface)**: UserD có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm, bao gồm cơ sở dữ liệu người dùng và các bộ sưu tập quản lý liên quan như *.system.profile và .system.js. Nó cũng bao gồm các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin.* và local.\*. hostManager có ít khả năng giám sát hơn so với clusterManager nhưng được ủy quyền để chạy một tập hợp rộng hơn các lệnh quản lý máy chủ, bao gồm setParameter, trafficRecord và auditConfigure.
* **Lịch trình (Schedule)**: Chúng ta giả định rằng hostManager có thể truy cập vào bề mặt của mình sau mỗi thao tác.
* **Chế độ xem (View)**: Nếu kiểm toán không được bật, chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của hostManager bao gồm dữ liệu liên quan đến QE là edb = (edc, esc, ecoc). Lệnh duy nhất được ủy quyền của hostManager có thể tiết lộ dữ liệu liên quan đến QE là trafficRecord, đầu ra của nó có thể bao gồm các bản gốc của các thao tác truy vấn. Do đó, chế độ xem của hostManager là của một kẻ thù (dbms,∞). Mặt khác, nếu kiểm toán được bật, thì hostManager có quyền truy cập vào auditLog. Tuy nhiên, chế độ xem của nó vẫn là của một kẻ thù (dbms,∞).

Ví dụ, giả sử chúng ta có một cụm MongoDB với ba cơ sở dữ liệu: “Database1”, “Database2” và “Database3”. Chúng ta có một người dùng tên là “UserF” với vai trò “hostManager”. UserF có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm, bao gồm “Database1”, “Database2”, “Database3” và các bộ sưu tập quản lý liên quan như *.system.profile và .system.js. Nó cũng bao gồm các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin.* và local.\*. UserF có thể chạy một số lượng lớn hơn và tập hợp các lệnh mạnh mẽ hơn so với clusterManager, bao gồm setParameter, trafficRecord và auditConfigure.

1. Vai trò sao lưu và phục hồi (Backup and Restoration Roles)

Vai trò của sao lưu ( backup roles)

Vai trò “backup” trong quản lý cụm MongoDB. Người dùng có vai trò này có thể sao lưu dữ liệu trên một cụm.

* **Bề mặt (Surface)**: UserD có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm, bao gồm cơ sở dữ liệu người dùng và các bộ sưu tập quản lý liên quan như *.system.profile và .system.js. Nó cũng bao gồm các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin.* và local.\*. Người dùng có vai trò backup có thể chạy một loạt các lệnh bao gồm serverStatus.
* **Lịch trình (Schedule)**: Chúng ta giả định rằng người dùng có vai trò backup có thể truy cập vào bề mặt của mình sau mỗi thao tác.
* **Chế độ xem (View)**: Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của người dùng có vai trò backup bao gồm dữ liệu liên quan đến QE là edb = (edc, esc, ecoc) và local.\* bao gồm local.oplog. Lệnh duy nhất được ủy quyền của người dùng có vai trò backup có thể chứa dữ liệu liên quan đến QE là serverStatus. Do đó, chế độ xem của người dùng có vai trò backup là của một kẻ thù (dbms,∞).

Giả sử chúng ta có một cụm MongoDB với ba cơ sở dữ liệu: “Database1”, “Database2” và “Database3”. Mỗi cơ sở dữ liệu này chứa các bảng dữ liệu khác nhau và các bộ sưu tập quản lý liên quan như \*.system.profile và \*.system.js.

Chúng ta có một người dùng tên là “UserG” với vai trò “backup”. UserG có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm. Điều này có nghĩa là UserG có thể xem tất cả các bảng và bản ghi trong “Database1”, “Database2” và “Database3”. Ngoài ra, UserG cũng có quyền truy cập vào các cơ sở dữ liệu nội bộ trên toàn hệ thống như config., admin. và local.\*.

Ví dụ, nếu UserG muốn sao lưu dữ liệu, họ có thể chạy một lệnh như mongodump để sao lưu toàn bộ cụm. Sau khi thực hiện lệnh này, UserG sẽ có một bản sao của tất cả dữ liệu trên cụm, bao gồm tất cả các bảng và bản ghi trong “Database1”, “Database2” và “Database3”, cũng như các cơ sở dữ liệu nội bộ như config., admin. và local.\*.

Tuy nhiên, mặc dù UserG có quyền truy cập vào tất cả dữ liệu này, họ không thể thay đổi hoặc chỉnh sửa nó. Vai trò “backup” chỉ cho phép UserG đọc dữ liệu, không cho phép ghi dữ liệu.

Vai trò phục hồi (Restore role)

Vai trò “restore” trong quản lý cụm MongoDB. Người dùng có vai trò này có thể khôi phục dữ liệu đã sao lưu vào một cụm.

* **Bề mặt (Surface)**: UserD có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm, bao gồm cơ sở dữ liệu người dùng và các bộ sưu tập quản lý liên quan như *.system.js và các cơ sở dữ liệu nội bộ trên toàn hệ thống như config.*, admin.\*. Khi tạo một vai trò khôi phục, có thể ủy quyền cho nó truy cập vào cơ sở dữ liệu *.system.profile và local.*. Tuy nhiên, chúng ta giả định rằng vai trò khôi phục không được phép làm như vậy. Người dùng có vai trò khôi phục có thể chạy một loạt các lệnh bao gồm $createIndex và $createCollection.
* **Lịch trình (Schedule)**: Chúng ta giả định rằng người dùng có vai trò khôi phục có thể truy cập vào bề mặt của mình sau mỗi thao tác.
* **Chế độ xem (View)**: Chỉ có cơ sở dữ liệu/bộ sưu tập trong bề mặt của người dùng có vai trò khôi phục bao gồm dữ liệu liên quan đến QE là edb = (edc, esc, ecoc). Không có lệnh nào được ủy quyền của người dùng có vai trò khôi phục chứa bất kỳ dữ liệu liên quan đến QE nào. Do đó, chế độ xem của người dùng có vai trò khôi phục là của một kẻ thù (edb, [n]).

**Ví dụ**: Giả sử chúng ta có một cụm MongoDB với ba cơ sở dữ liệu: “Database1”, “Database2” và “Database3”. Mỗi cơ sở dữ liệu này chứa các bảng dữ liệu khác nhau và các bộ sưu tập quản lý liên quan như \*.system.js.

Chúng ta có một người dùng tên là “UserH” với vai trò “restore”. UserH có quyền truy cập vào tất cả các cơ sở dữ liệu trên cụm. Điều này có nghĩa là UserH có thể xem tất cả các bảng và bản ghi trong “Database1”, “Database2” và “Database3”. Ngoài ra, UserH cũng có quyền truy cập vào các cơ sở dữ liệu nội bộ trên toàn hệ thống như config.\* và admin.\*.

Ví dụ, nếu UserH muốn khôi phục dữ liệu từ một bản sao lưu, họ có thể chạy một lệnh như mongorestore để khôi phục toàn bộ cụm từ bản sao lưu. Sau khi thực hiện lệnh này, UserH sẽ có một bản sao của tất cả dữ liệu trên cụm, bao gồm tất cả các bảng và bản ghi trong “Database1”, “Database2” và “Database3”, cũng như các cơ sở dữ liệu nội bộ như config.\* và admin.\*.

Tuy nhiên, mặc dù UserH có quyền truy cập vào tất cả dữ liệu này, họ không thể thay đổi hoặc chỉnh sửa nó. Vai trò “restore” chỉ cho phép UserH đọc dữ liệu, không cho phép ghi dữ liệu.

1. Nguyên tắc
2. Lập kế hoạch nén ( Scheduling Compactions )

* **Compaction** là một hoạt động QE giúp giảm kích thước của cơ sở dữ liệu được mã hóa bằng cách loại bỏ dữ liệu lỗi thời từ ecoc và esc. Sau quá trình compaction, ecoc được đặt lại thành một bộ sưu tập trống và kích thước của esc giảm đi.
* Trước khi compaction, ecoc và esc của một bộ sưu tập sẽ mỗi người ví dụ sẽ giữ 1,000 meta documents. Sau compaction, esc giảm xuống còn 3 meta documents và ecoc trống. Điều này giúp tiết kiệm khoảng 99.85% dung lượng lưu trữ



Trong đó:

* encfields là số lượng trường được mã hóa trong bộ sưu tập.
* docinsertsi là số lượng tài liệu được chèn giữa lần compaction thứ (i-1) và lần compaction thứ i.
* valinsertsi là số lượng cặp trường/giá trị duy nhất được chèn giữa lần compaction thứ (i-1) và lần compaction thứ i.
* t là số lượng meta documents mà muốn giảm sau mỗi lần compaction.

Công thức này cho thấy khi nào nên thực hiện compaction. Nếu kết quả của biểu thức encfields · docinsertsi - valinsertsi lớn hơn hoặc bằng t, nên thực hiện compaction.

Ví dụ, giả sử có một bộ sưu tập với 2 trường được mã hóa là “name” và “address”. đã chèn 100 tài liệu mới vào bộ sưu tập giữa hai lần compaction liên tiếp, và trong số đó có 50 cặp trường/giá trị duy nhất. muốn giảm ít nhất 100 meta documents sau mỗi lần compaction. Khi đó, sẽ thay các giá trị này vào công thức:

encfields · docinsertsi - valinsertsi = 2 · 100 - 50 = 150

Vì 150 lớn hơn 100, nên thực hiện compaction.

Ví dụ, giả sử có một cơ sở dữ liệu với 1,000 tài liệu, trong đó:

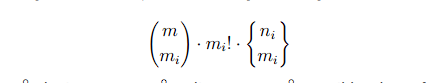
* 500 tài liệu có trường “name” với giá trị là “Alice”
* 250 tài liệu có trường “name” với giá trị là “Bob”
* 250 tài liệu có trường “name” với giá trị là “Eve”

Trước khi thực hiện compaction, cả ecoc và esc đều chứa 1,000 meta documents. Mỗi meta document tương ứng với một tài liệu trong cơ sở dữ liệu.

Sau khi thực hiện compaction, esc chỉ còn 3 meta documents, tương ứng với 3 giá trị duy nhất của trường “name”: “Alice”, “Bob” và “Eve”. Và ecoc trở thành một bộ sưu tập trống.

**Sercurity:**

valinsertsi và compactions: Khi thực hiện compaction, số lượng cặp trường/giá trị được mã hóa QE duy nhất kể từ lần compaction cuối cùng (valinsertsi) sẽ được tiết lộ. Trong ví dụ được đưa ra, #valinserts = 3, có nghĩa là kẻ tấn công biết rằng có 3 cặp trường/giá trị duy nhất đã được chèn kể từ lần compaction cuối cùng.

**Thiếu thông tin phụ trợ (auxiliary information)**: Nếu không có thông tin phụ trợ, kẻ tấn công chỉ biết rằng các tài liệu được chèn vào là một trong số các khả năng của

Trong đó:

* ni là số lượng tài liệu được chèn (docinsertsi)
* mi là số lượng cặp trường/giá trị duy nhất được chèn (valinsertsi)
* m là số lượng giá trị có thể mà một trường có thể nhận (possiblevals)
* (ni/mi​) là Stirling numbers of the second kind.

1. Đặt hệ số tranh chấp (Setting the Contention Factor)

**Contention** (sự cạnh tranh): Đây là hiện tượng xảy ra khi nhiều client cố gắng thực hiện các thao tác trên cùng một dữ liệu. Ví dụ, contention có thể xảy ra khi nhiều client cố gắng chèn các tài liệu có cùng giá trị id hoặc cập nhật các tài liệu có cùng cặp trường/giá trị.

**Contention Factor (hệ số cạnh tranh)**: Đối với mỗi trường được mã hóa QE, QE định nghĩa một tham số

cff≥0

được gọi là hệ số cạnh tranh. Việc tăng hệ số cạnh tranh của một trường sẽ làm tăng hiệu suất của các thao tác chèn và cập nhật trên trường đó, nhưng sẽ làm giảm hiệu suất của các thao tác tìm kiếm.

Mặc định, tất cả các hệ số cạnh tranh đều được đặt là 8. Điều này dựa trên nhiều thí nghiệm và cho thấy rằng nó cung cấp hiệu suất tốt cho nhiều loại công việc khác nhau.

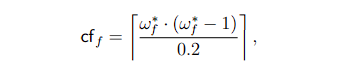
**Điều chỉnh hệ số cạnh tranh (Changing the contention factor)**

Hệ số cạnh tranh mặc định thích hợp cho hầu hết các cài đặt nhưng có thể được tinh chỉnh theo đặc điểm của công việc. Nếu công việc có xu hướng có số lượng rất lớn các thao tác chèn hoặc cập nhật đồng thời trên một trường f, thì có thể hợp lý khi tăng hệ số cạnh tranh để tăng hiệu suất chèn hoặc cập nhật. Ngược lại, nếu công việc có xu hướng có số lượng rất nhỏ các thao tác chèn hoặc cập nhật đồng thời, thì giảm hệ số cạnh tranh có thể giúp cải thiện hiệu suất đọc.

**Công thức điều chỉnh hệ số cạnh tranh**:

Giả sử ωf​

là ước lượng tổng số lượng các thao tác chèn hoặc cập nhật đồng thời trên một trường f trong một khoảng thời gian nhỏ, ví dụ, 30ms. Hệ số cạnh tranh cff​

có thể được đặt như sau:

trong đó:

Nếu số lượng các thao tác chèn/cập nhật đồng thời trên một trường không được biết, ωf​ có thể được đặt bằng số lượng lõi ảo của máy chủ.

**Ảnh hưởng đến hiệu quả tìm kiếm**: Như đã đề cập ở trên, việc tăng hệ số cạnh tranh có thể làm giảm hiệu quả của các thao tác tìm kiếm. Cụ thể hơn, hiệu quả tìm kiếm được xác định như sau:

timefind=O(cff​⋅log(metadocs(esc))+∣R∣)

trong đó R là tập hợp các tài liệu được trả về bởi thao tác tìm kiếm. Lưu ý rằng nếu R nhỏ, hệ số cạnh tranh sẽ ảnh hưởng nhiều hơn đến hiệu suất đọc. Cụ thể, việc sử dụng các giá trị lớn của

cff​

có thể làm giảm đáng kể hiệu suất đọc.

Ví dụ :

Giả sử có một cơ sở dữ liệu với một trường f và muốn tối ưu hóa hiệu suất cho một công việc cụ thể. Công việc này có xu hướng có một lượng lớn các thao tác chèn hoặc cập nhật đồng thời trên trường f.

**Tính toán** :**ωf​**

Đầu tiên, cần ước lượng tổng số lượng các thao tác chèn hoặc cập nhật đồng thời trên trường f trong một khoảng thời gian nhỏ, ví dụ, 30ms. Giả sử ωf​=100

**Tính toán ωf∗​**

: Tiếp theo, cần tính toán

ωf∗​=⌈possiblevalsωf​​⌉

.Giả sử possiblevals (số lượng giá trị có thể có của trường f) là 50, vậy thì

ωf∗​=⌈100/50​⌉=2

**Điều chỉnh hệ số cạnh tranh**: Cuối cùng, có thể điều chỉnh hệ số cạnh tranh cff​

bằng cách sử dụng công thức sau:

cff​=ωf∗​⋅(ωf∗​−1)/0.2​=2\*(2−1)/(0.2)​=10

Trong trường hợp này, hệ số cạnh tranh lớn hơn giá trị mặc định là 8 và sẽ dẫn đến việc giảm hiệu suất đọc mà không ảnh hưởng đến hiệu suất chèn và cập nhật.

**Ảnh hưởng đến sự rò rỉ thông tin**: Hệ số cạnh tranh không bao giờ nên được đặt dựa trên thông tin bí mật / nhạy cảm. Ví dụ, nó không nên được đặt dựa trên tần suất của các cặp trường / giá trị.

**Ví dụ về việc điều chỉnh hệ số cạnh tranh**: Xem xét trường hợp valinserts = 1000 và ω = 100. Dựa trên phương pháp đã trình bày ở trên, người ta có thể đặt cff = 1. Ngược lại, giả sử một người nhận thấy rằng đối với một số trường f, có khoảng 20 giá trị thường xuất hiện rất thường xuyên trong bộ sưu tập và do đó, đặt cff > 1. Lựa chọn này của cff có thể tiết lộ thêm thông tin cho một kẻ thù đa chụp ảnh vì hệ số cạnh tranh được tính toán dựa trên phân phối chèn / cập nhật.

**Khuyến nghị về việc điều chỉnh hệ số cạnh tranh**: Do điều này, đề nghị giữ hệ số cạnh tranh ở giá trị mặc định và nếu nó phải được điều chỉnh, thì nó nên được đặt theo cách tiếp cận đã trình bày ở trên.

1. Xử lý các giá trị có độ dài nhạy cảm ( andling Values with Sensitive Lengths)

**Plaintext recovery through length.**

Vấn đề: Khi mã hóa các giá trị có thể truy vấn, độ dài của các giá trị được tiết lộ cho kẻ thù. Điều này có thể giúp kẻ thù khôi phục lại các giá trị bản rõ nếu họ biết trước một số giá trị có thể có và độ dài của chúng.

Ví dụ: Đoạn văn đưa ra một ví dụ về một trường có 3 giá trị có thể có v1, v2 và v3 với các độ dài khác nhau. Khi mã hóa chúng bằng AES256-CBC, ta thu được 3 bản mã ct1, ct2 và ct3 với các độ dài gần bằng độ dài của các giá trị bản rõ. Từ đó, kẻ thù có thể suy ra được ct1 là mã hóa của v1, ct2 là mã hóa của v2 và ct3 là mã hóa của v3.

Giải pháp: Đoạn văn đề xuất hai cách tiếp cận phía khách hàng để tránh vấn đề trên. Một là thêm đệm ngẫu nhiên vào các giá trị bản rõ để làm cho chúng có cùng độ dài. Hai là sử dụng một hàm băm để biến đổi các giá trị bản rõ trước khi mã hóa. Đoạn văn cũng lưu ý rằng những giải pháp này không được cung cấp bởi các trình điều khiển MongoDB và phải được triển khai bởi ứng dụng.

**Padding**

Giải pháp: Thêm đệm vào mỗi giá trị của trường để làm cho nó có độ dài bằng với giá trị có độ dài lớn nhất trong miền của trường trước khi mã hóa. Ví dụ, giả sử một trường f có thể nhận các giá trị trong tập V = {v1, · · · , vn} và giả sử vn là giá trị có độ dài lớn nhất. Trước khi mã hóa tập hợp với QE, người ta nên thêm đệm vào mỗi giá trị f v sao cho |v| = |vn|, trong đó |v| là độ dài của giá trị v.

Nhược điểm: Cách tiếp cận này làm tăng kích thước của tập hợp.

**Mapping**

Giải pháp: Ánh xạ các giá trị của trường f sang một tập hợp khác của các giá trị có độ dài cố định trước khi truyền tập hợp đó cho QE. Cụ thể hơn, giả sử một trường f có miền V = {v1, · · · , vn} trong đó các giá trị có độ dài khác nhau. Với cách tiếp cận này, người ta sẽ tạo một miền mới W = {1, · · · , n} của các số có độ dài cố định và một ánh xạ T : V → W để ánh xạ các giá trị trong V sang các giá trị trong W. Ánh xạ T này có thể được biểu diễn dưới dạng một bảng ví dụ. Ở đây, ta giả sử rằng n không quá lớn và rằng các số từ 1 đến n được biểu diễn bằng một số bit cố định, ví dụ log2​(n) bit. Trước khi mã hóa tập hợp với QE, người ta sẽ thay thế mỗi giá trị f v trong tập hợp bằng w = T(v).

Ưu điểm: Khác với cách tiếp cận thêm đệm, kích thước của tập hợp không tăng nhiều.

Nhược điểm: Đổi lại là bảng T cần được lưu trữ và quản lý phía khách hàng.

1. Sử dụng đường ống tổng hợp (Using the Aggregation Pipeline)

Aggregation pipeline và mã hóa có thể truy vấn (queryable encryption) của MongoDB. Đây là hai kỹ thuật quan trọng trong việc xử lý và bảo mật dữ liệu trong MongoDB, một hệ quản trị cơ sở dữ liệu không quan hệ (NoSQL).

Khung trình tổng hợp là một kỹ thuật cho phép xây dựng các truy vấn dưới dạng một chuỗi các giai đoạn, mỗi giai đoạn xử lý các tài liệu trong một tập hợp và truyền kết quả làm đầu vào cho các giai đoạn tiếp theo. Có thể sử dụng khung trình tổng hợp để thực hiện các thao tác như lọc, nhóm, sắp xếp, thống kê, biến đổi dữ liệu theo nhiều cách khác nhau.

Mã hóa có thể truy vấn là một kỹ thuật cho phép mã hóa các trường nhạy cảm trong các tài liệu bằng mã hóa ngẫu nhiên, trong khi vẫn có thể truy vấn các trường được mã hóa. Với mã hóa có thể truy vấn, một giá trị bản rõ luôn mã hóa thành một bản mã khác nhau, trong khi vẫn có thể truy vấn được. Có thể sử dụng mã hóa có thể truy vấn để bảo vệ dữ liệu của khỏi các cuộc tấn công từ bên ngoài hoặc bên trong.

Để sử dụng khung trình tổng hợp và mã hóa có thể truy vấn của MongoDB, cần phải chú ý đến một số điểm sau:

* Khi QE được sử dụng ở chế độ tự động (automatic mode), các trình điều khiển (drivers) có quyền truy cập vào một mô-đun phân tích truy vấn phía khách hàng (client-side query analysis module) sẽ hạn chế các giai đoạn có thể được sử dụng trong một khung trình tổng hợp. Cụ thể, mô-đun phân tích truy vấn sẽ không cho phép bất kỳ giai đoạn nào thay đổi cơ sở dữ liệu. Điều này sẽ đảm bảo rằng không có khung trình tổng hợp nào có thể thay đổi hồ sơ rò rỉ (leakage profile) của QE đối với các kẻ thù chụp ảnh nhanh (snapshot adversaries).
* Khi QE được sử dụng ở chế độ rõ ràng (explicit mode), các trình điều khiển không có quyền truy cập vào mô-đun phân tích truy vấn phía khách hàng. Điều này có nghĩa là bất kỳ giai đoạn và khung trình tổng hợp nào cũng có thể được thực hiện, có thể thay đổi hồ sơ rò rỉ của QE. Do đó, khi sử dụng QE với mã hóa rõ ràng, chúng tôi khuyên nên tránh bất kỳ giai đoạn nào có thể thay đổi cơ sở dữ liệu.

1. Bổ sung

Mã hóa Một Phần và Lựa Chọn Trường:

QE cho phép người dùng chọn các trường cụ thể trong tài liệu cần được mã hóa.

Gợi ý ban đầu có thể là chỉ mã hóa các trường "nhạy cảm", nhưng văn bản nhấn mạnh rằng quyết định này không đơn giản.

Cần xem xét sự tương quan giữa các trường. Nếu biết một trường có thể cung cấp thông tin về một trường khác, thì việc chỉ mã hóa một trường có thể không đảm bảo an toàn.

Đưa ra ví dụ với các trường "city" và "state". Mã hóa chỉ trường "state" có thể không đủ nếu trường "city" có thể tiết lộ thông tin về trạng thái.

Định Nghĩa Schema trong QE:

Người dùng xác định một schema khi sử dụng QE, chỉ định các trường nào cần được mã hóa, loại truy vấn cần hỗ trợ, hệ số cạnh tranh và khóa mã hóa cần sử dụng.

Schema có thể được lưu trữ ở phía máy chủ hoặc ở phía máy khách.

Cảnh báo rằng hiện tại QE không có cơ chế để xác minh tính toàn vẹn của một schema lưu trữ ở phía máy chủ. Nếu máy chủ bị xâm phạm, kẻ tấn công có thể làm thay đổi schema, dẫn đến việc gửi các giá trị không mã hóa thay vì giá trị được mã hóa.

Gợi Ý về Lưu Trữ Schema:

Do rủi ro có thể máy chủ bị xâm phạm, gợi ý là nên giữ schema QE ở phía máy khách trong tất cả các trường hợp.

Bảo Mật và Ghi Chú về Log Kiểm Tra:

MongoDB Enterprise Advanced (EA) cung cấp khả năng kiểm tra, khi được kích hoạt, theo dõi nhiều sự kiện hệ thống và người dùng.

Việc kích hoạt log kiểm tra có thể cho phép một số vai trò, đặc biệt là "hostManager", truy cập chi tiết về tương tác giữa một khách hàng QE và máy chủ.

Lưu ý rằng thông tin có sẵn trong log kiểm tra không chỉ có sẵn cho vai trò hostManager mà còn có thể được lấy từ lệnh $trafficRecord.

**Demo cơ bản**

Giả sử có một tập hợp (collection) tên là employees chứa các tài liệu (document) về thông tin nhân viên của một công ty. Mỗi tài liệu có các trường (field) như name, age, salary, department, position, email, phone. muốn mã hóa các trường nhạy cảm như salary, email, phone để bảo vệ dữ liệu của . cũng muốn truy vấn các tài liệu theo các tiêu chí khác nhau, ví dụ như lọc theo department, sắp xếp theo salary, nhóm theo position, tính trung bình age của mỗi nhóm, v.v.

Để làm được điều này, cần sử dụng mã hóa có thể truy vấn (QE) và khung trình tổng hợp (aggregation pipeline) của MongoDB. có thể tham khảo các bước sau:

Bước 1: Tạo một khóa mã hóa (encryption key) để sử dụng cho QE. có thể sử dụng công cụ mongocryptd để tạo một khóa ngẫu nhiên hoặc tự tạo một khóa bằng cách sử dụng thuật toán AES-2561.

Bước 2: Tạo một kết nối (connection) đến cơ sở dữ liệu MongoDB của bằng cách sử dụng một trình điều khiển (driver) hỗ trợ QE, ví dụ như Python, Java, Node.js, v.v. cần cung cấp khóa mã hóa, tên cơ sở dữ liệu, tên tập hợp và các trường cần mã hóa cho trình điều khiển2.

Bước 3: Chèn (insert) các tài liệu vào tập hợp employees bằng cách sử dụng phương thức insert\_one hoặc insert\_many của trình điều khiển. Các trường được chỉ định để mã hóa sẽ được mã hóa tự động bằng QE trước khi được lưu trữ vào cơ sở dữ liệu2.

Bước 4: Truy vấn (query) các tài liệu trong tập hợp employees bằng cách sử dụng phương thức aggregate của trình điều khiển. có thể xây dựng một khung trình tổng hợp (aggregation pipeline) bao gồm nhiều giai đoạn (stage) khác nhau để xử lý và biến đổi dữ liệu theo ý muốn3. Các trường được mã hóa sẽ được giải mã tự động bằng QE trước khi được trả về cho 2.

Dưới đây là một ví dụ về cách sử dụng Python để thực hiện các bước

Giả sử có một tập hợp (collection) tên là employees chứa các tài liệu (document) về thông tin nhân viên của một công ty. Mỗi tài liệu có các trường (field) như name, age, gender, salary, department, position, address, phone, email. muốn bảo mật một số trường nhạy cảm như salary, address, phone, email bằng cách mã hóa chúng bằng mã hóa có thể truy vấn (QE). cũng muốn thực hiện một số truy vấn trên các trường được mã hóa, ví dụ như tìm kiếm nhân viên có mức lương bằng hoặc cao hơn một giá trị nào đó, hoặc tìm kiếm nhân viên có địa chỉ ở một khu vực nào đó.

Để làm được điều này, cần làm các bước sau:

Bước 1: Tạo một file JSON để định nghĩa cấu hình mã hóa cho tập hợp employees. File này sẽ chỉ định các trường nào sẽ được mã hóa, kiểu mã hóa, và khóa mã hóa. Ví dụ:

*{*

*"bsonType": "object",*

*"properties": {*

*"name": {*

*"encrypt": {*

*"bsonType": "string",*

*"algorithm": "AEAD\_AES\_256\_CBC\_HMAC\_SHA\_512-Deterministic",*

*"keyId": [*

*{*

*"$binary": {*

*"base64": "AAAAAAAAAAAAAAAAAAAAAA==",*

*"subType": "04"*

*}*

*}*

*]*

*}*

*},*

*"age": {*

*"bsonType": "int"*

*},*

*"gender": {*

*"bsonType": "string"*

*},*

*"salary": {*

*"encrypt": {*

*"bsonType": "int",*

*"algorithm": "AEAD\_AES\_256\_CBC\_HMAC\_SHA\_512-Random",*

*"keyId": [*

*{*

*"$binary": {*

*"base64": "AAAAAAAAAAAAAAAAAAAAAA==",*

*"subType": "04"*

*}*

*}*

*]*

*}*

*},*

*"department": {*

*"bsonType": "string"*

*},*

*"position": {*

*"bsonType": "string"*

*},*

*"address": {*

*"encrypt": {*

*"bsonType": "string",*

*"algorithm": "AEAD\_AES\_256\_CBC\_HMAC\_SHA\_512-Random",*

*"keyId": [*

*{*

*"$binary": {*

*"base64": "AAAAAAAAAAAAAAAAAAAAAA==",*

*"subType": "04"*

*}*

*}*

*]*

*}*

*},*

*"phone": {*

*"encrypt": {*

*"bsonType": "string",*

*"algorithm": "AEAD\_AES\_256\_CBC\_HMAC\_SHA\_512-Random",*

*"keyId": [*

*{*

*"$binary": {*

*"base64": "AAAAAAAAAAAAAAAAAAAAAA==",*

*"subType": "04"*

*}*

*}*

*]*

*}*

*},*

*"email": {*

*"encrypt": {*

*"bsonType": "string",*

*"algorithm": "AEAD\_AES\_256\_CBC\_HMAC\_SHA\_512-Random",*

*"keyId": [*

*{*

*"$binary": {*

*"base64": "AAAAAAAAAAAAAAAAAAAAAA==",*

*"subType": "04"*

*}*

*}*

*]*

*}*

*}*

*}*

*}*

Trong file này, có thể thấy rằng các trường name, salary, address, phone, email sẽ được mã hóa bằng QE. Các trường khác sẽ không được mã hóa. cũng có thể thấy rằng các trường được mã hóa sử dụng hai kiểu thuật toán khác nhau: AEAD\_AES\_256\_CBC\_HMAC\_SHA\_512-Deterministic và AEAD\_AES\_256\_CBC\_HMAC\_SHA\_512-Random. Kiểu thuật toán xác định cách mã hóa và truy vấn các giá trị của trường. Với kiểu Deterministic, một giá trị bản rõ sẽ luôn mã hóa thành một bản mã cố định, và có thể truy vấn bằng toán tử bằng ($eq) hoặc nằm trong ($in). Với kiểu Random, một giá trị bản rõ sẽ mã hóa thành một bản mã ngẫu nhiên, và có thể truy vấn bằng các toán tử so sánh ($gt, $lt, $gte, $lte). cũng cần chỉ định khóa mã hóa cho mỗi trường, trong ví dụ này là một khóa có giá trị AAAAAAAAAAAAAAAAAAAAAA==1.

Bước 2: Tạo một client-side encryption object để mã hóa và giải mã các trường được chỉ định. cần cung cấp một số thông tin như URI của cơ sở dữ liệu, tên của tập hợp chứa khóa mã hóa, file JSON định nghĩa cấu hình mã hóa, và một số tùy chọn khác. Ví dụ:

*from pymongo import MongoClient*

*from pymongo.encryption import ClientEncryption*

*from pymongo.encryption\_options import AutoEncryptionOpts*

*client = MongoClient("mongodb://localhost:27017/")*

*key\_vault = client["key\_vault"]["data\_keys"]*

*encryption\_schema = open("encryption\_schema.json").read()*

*encryption\_opts = AutoEncryptionOpts(*

*kms\_providers={"local": {"key": b"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="}},*

*key\_vault\_namespace="key\_vault.data\_keys",*

*schema\_map={"employees": encryption\_schema}*

*)*

*encrypted\_client = MongoClient("mongodb://localhost:27017/", auto\_encryption\_opts=encryption\_opts)*

Trong đoạn code này, tạo một client bình thường để kết nối với cơ sở dữ liệu, một key vault để lưu trữ khóa mã hóa, và một encrypted client để thực hiện các thao tác mã hóa và giải mã. cũng cần cung cấp các nhà cung cấp KMS (Key Management Service) để quản lý khóa mã hóa, trong ví dụ này là một khóa cục bộ. cũng cần chỉ định không gian tên (namespace) của key vault và bản đồ lược đồ (schema map) của tập hợp employees2.

Bước 3: Thêm các tài liệu vào tập hợp employees bằng encrypted client. Khi thêm các tài liệu vào tập hợp, các trường được chỉ định sẽ được mã hóa tự động bằng QE. Ví dụ:

*employees = encrypted\_client["company"]["employees"]*

*employees.insert\_one({*

*"name": "Alice",*

*"age": 25,*

*"gender": "female",*

*"salary": 5000,*

*"department": "IT",*

*"position": "developer",*

*"address": "123 Main Street",*

*"phone": "0123456789",*

*"email": "alice@example.com"*

*})*

*employees.insert\_one({*

*"name": "Bob",*

*"age": 30,*

*"gender": "male",*

*"salary": 6000,*

*"department": "HR",*

*"position": "manager",*

*"address": "456 High Road",*

*"phone": "0987654321",*

*"email": "bob@example.com"*

*})*

Trong đoạn code này, thêm hai tài liệu vào tập hợp employees, mỗi tài liệu chứa thông tin của một nhân viên. có thể kiểm tra trong cơ sở dữ liệu rằng các trường name, salary, address, phone, email sẽ được mã hóa bằng một đoạn mã nhị phân.

Bước 4: Thực hiện các truy vấn trên tập hợp employees bằng cách sử dụng khung trình tổng hợp. Khung trình tổng hợp là một kỹ thuật cho phép xây dựng các truy vấn dưới dạng một chuỗi các giai đoạn, mỗi giai đoạn xử lý các tài liệu trong một tập hợp và truyền kết quả làm đầu vào cho các giai đoạn tiếp theo1. có thể sử dụng khung trình tổng hợp để thực hiện các thao tác như lọc, nhóm, sắp xếp, thống kê, biến đổi dữ liệu theo nhiều cách khác nhau1.

Để sử dụng khung trình tổng hợp với tập hợp employees, cần làm các bước sau:

Bước 4.1: Tạo một biến để lưu trữ khung trình tổng hợp. có thể sử dụng phương thức aggregate() của tập hợp employees để tạo khung trình tổng hợp. cần truyền vào một mảng các giai đoạn (stage) cho phương thức này. Mỗi giai đoạn là một đối tượng JSON có dạng {<tên giai đoạn>: <tham số>}. Ví dụ:

Python

AI-generated code. Review and use carefully. More info on FAQ.

pipeline = employees.aggregate([

# Đây là các giai đoạn của khung trình tổng hợp

])

Bước 4.2: Thêm các giai đoạn vào khung trình tổng hợp. có thể sử dụng nhiều giai đoạn khác nhau để xử lý dữ liệu theo ý muốn. Một số giai đoạn phổ biến là:

*$match*: Lọc các tài liệu theo một tiêu chí nào đó. Ví dụ: *{"$match": {"salary": {"$gte": 5000}}}* sẽ lọc ra các tài liệu có trường salary lớn hơn hoặc bằng 5000. có thể sử dụng các toán tử truy vấn của MongoDB để xây dựng tiêu chí lọc.

*$project*: Chọn các trường mà muốn hiển thị trong kết quả. Ví dụ: *{"$project": {"name": 1, "salary": 1, "\_id": 0}}* sẽ chỉ hiển thị các trường name và salary trong kết quả, và loại bỏ trường \_id. có thể sử dụng giá trị 1 hoặc 0 để chỉ định trường nào được hiển thị hoặc ẩn.

*$group*: Nhóm các tài liệu theo một trường nào đó và thực hiện các thống kê trên nhóm. Ví dụ*: {"$group": {"\_id": "$department", "total\_salary": {"$sum": "$salary"}}}* sẽ nhóm các tài liệu theo trường department và tính tổng lương của mỗi nhóm. có thể sử dụng các toán tử nhóm của MongoDB để thực hiện các thống kê.

*$sort*: Sắp xếp các tài liệu theo một trường nào đó. Ví dụ: *{"$sort": {"salary": -1}}* sẽ sắp xếp các tài liệu theo trường salary giảm dần. có thể sử dụng giá trị 1 hoặc -1 để chỉ định thứ tự tăng dần hoặc giảm dần.

Bước 4.3: Chạy khung trình tổng hợp và xem kết quả. có thể sử dụng phương thức next() hoặc toArray() của biến pipeline để lấy kết quả của khung trình tổng hợp. Ví dụ:

*result = pipeline.next()*

*result = pipeline.toArray()*

Kết quả sẽ là một tài liệu hoặc một mảng các tài liệu, tùy thuộc vào số lượng giai đoạn và thao tác của . có thể xem, in ra, hoặc lưu trữ kết quả theo ý muốn.