BAN CƠ YẾU CHÍNH PHỦ

BÁO CÁO ĐỀ TÀI NHÁNH

"NGHIÊN CỨU, XÂY DỰNG GIẢI PHÁP BẢO MẬT THÔNG TIN TRONG THƯƠNG MẠI ĐIỆN TỬ"

SẢN PHẨM SỐ 3: AN TOÀN THÔNG TIN CHO CƠ SỞ DỮ LIỆU

Thuộc đề tài : "Nghiên cứu một số vấn đề kỹ thuật, công nghệ chủ yếu trong thương mại điện tử và triển khai thử nghiệm – Mã số KC.01.05"

6095-4
14/9/2006

Hà nội, tháng 9 năm 2004

NỘI DUNG

TỔNG QUAN VỀ
AN TOÀN THÔNG TIN TRONG CƠ SỞ DỮ LIỆU

1 Giới thiệu

Sự phát triển lớn mạnh của công nghệ thông tin trong những năm qua đã dẫn đến sử dụng rộng rãi hệ thống máy tính trong mọi tổ chức cá nhân và công cộng, chẳng hạn như ngân hàng, trường học, tổ chức dịch vụ và sản xuất. Độ tin cậy của phần cứng, phần mềm ngày một được nâng cao cùng với việc liên tục giảm giá, tăng kỹ năng chuyên môn của các chuyên viên thông tin và sự sẵn sàng của các công cụ trợ giúp đã góp phần khuyến khích việc sử dụng dịch vụ máy tính một cách rộng rãi. Vì vậy, dữ liệu được lưu giữ và quản lý trong các hệ thống máy tính nhiều hơn. Cơ sở dữ liệu sử dụng các hệ quản trị cơ sở dữ liệu đã đáp ứng được các yêu cầu về lưu giữ và quản lý dữ liệu.

Nhiều phương pháp luận thiết kế cơ sở dữ liệu đã được phát triển nhằm hỗ trợ các yêu cầu thông tin khác nhau và các môi trường làm việc của ứng dụng. Các mô hình dữ liệu khái niệm và lôgíc đã được nghiên cứu, cùng với những ngôn ngữ thích hợp, các công cụ định nghĩa dữ liệu, thao tác và hỏi đáp dữ liệu. Mục tiêu là đưa ra các DBMS có khả năng quản trị và khai thác dữ liệu tốt.

Một đặc điểm cơ bản của DBMS là khả năng quản lý đồng thời nhiều giao diện ứng dụng. Mỗi ứng dụng có một cái nhìn thuần nhất về cơ sở dữ liệu, có nghĩa là có cảm giác chỉ mình nó đang khai thác cơ sở dữ liệu. Đây là một yêu cầu hết sức quan trọng đối với các DBMS, ví dụ cơ sở dữ liệu của ngân hàng với các khách hàng trực tuyến của nó; hoặc cơ sở dữ liệu của các hãng hàng không với việc đặt vé trước.

Xử lý phân tán đã góp phần phát triển và tự động hoá các hệ thống thông tin. Ngày nay, đơn vị xử lý thông tin của các tổ chức và các chi nhánh ở xa của nó có thể giao tiếp với nhau một cách nhanh chóng thông qua các mạng máy tính, vì vậy cho phép truyền tải rất nhanh các khối dữ liệu lớn.

Việc sử dụng rộng rãi các cơ sở dữ liệu phân tán và tập trung đã đặt ra nhiều yêu cầu nhằm đảm bảo các chức năng thương mại và an toàn dữ liệu. Trong thực tế, các sự cố trong môi trường cơ sở dữ liệu không chỉ ảnh hưởng đến từng người sử dụng

hoặc ứng dụng, mà còn ảnh hưởng tới toàn bộ hệ thống thông tin. Các tiến bộ trong kỹ thuật xử lý thông tin (các công cụ và ngôn ngữ) đã đơn giản hoá giao diện giữa người và máy phục vụ cho việc tạo ra các cơ sở dữ liệu đáp ứng được cho nhiều dạng người dùng khác nhau; Vì vậy đã nảy sinh thêm nhiều vấn đề về an toàn. Trong các hệ thống thông tin, máy tính, kỹ thuật, công cụ và các thủ tục an toàn đóng vai trò thiết yếu, đảm bảo tính liên tục và tin cậy của hệ thống, bảo vệ dữ liệu và các chương trình không bị xâm nhập, sửa đổi, đánh cắp và tiết lộ thông tin trái phép.

An toàn thông tin trong cơ sở dữ liệu

An toàn thông tin trong cơ sở dữ liệu bao gồm 3 yếu tố chính: tính bí mật, toàn vẹn và sẵn sàng. Trong tài liệu này, các thuật ngữ như gán quyền, bảo vệ và an toàn sẽ được sử dụng để diễn đạt cùng một nội dung trong các ngữ cảnh khác nhau. Chính xác hơn, thuật ngữ *gán quyền* được sử dụng trong các hệ thống cơ sở dữ liệu, thuật ngữ *bảo vệ* thường sử dụng khi nói về hệ điều hành, còn thuật ngữ *an toàn* được sử dụng chung.

Bảo mật là ngăn chặn, phát hiện và xác định những tiếp cận thông tin trái phép. Nói chung, bảo mật là bảo vệ dữ liệu trong các môi trường cần bảo mật cao, ví dụ như các trung tâm quân sự hay kinh tế quan trọng. Tính riêng tư (*privacy)* là thuật ngữ chỉ ra quyền của một cá nhân, một nhóm người, hoặc một tổ chức đối với các thông tin, tài nguyên nào đó. Tính riêng tư được luật pháp của nhiều quốc gia bảo đảm. Bí mật là yếu tố quan trọng nhất để đảm bảo an toàn trong các môi trường, cả quân sự lẫn thương mại. Đảm bảo tính toàn vẹn có nghĩa là ngăn chặn, phát hiện và xác định các sửa đổi thông tin trái phép. Đảm bảo tính sẵn sàng có nghĩa là ngăn chặn, phát hiện và xác định các từ chối truy nhập chính đáng vào các dịch vụ mà hệ thống cung cấp.

2. Một số khái niệm CSDL

Cơ sở dữ liệu là một tập hợp dữ liệu không nhất thiết đồng nhất, có quan hệ với nhau về mặt lôgíc và được phân bố trên một mạng máy tính.

Hệ thống phần mềm cho phép quản lý, thao tác trên cơ sở dữ liệu, tạo ra sự trong suốt phân tán với người dùng gọi là hệ quản trị cơ sở dữ liệu (DBMS).

Trong thiết kế cơ sở dữ liệu, chúng ta cần phân biệt pha quan niệm và pha lôgíc. Các mô hình quan niệm và lôgíc tương ứng thường dùng để mô tả cấu trúc của cơ sở dữ liệu. Trong các mô hình này, mô hình lôgíc phụ thuộc vào hệ quản trị cơ sở dữ liệu, còn mô hình quan niệm thì độc lập với hệ quản trị cơ sở dữ liệu. Mô hình quan hệ thực thể là một trong các mô hình quan niệm phổ biến nhất, được xây dựng dựa trên khái niệm thực thể. Thực thể được xem như là lớp các đối tượng của thế giới hiện thực được mô tả bên trong cơ sở dữ liệu và quan hệ mô tả mối liên hệ giữa hai hay nhiều thực thể.

Trong quá trình thiết kế lôgíc, lược đồ khái niệm được chuyển sang lược đồ lôgíc, mô tả dữ liệu theo mô hình lôgíc do DBMS cung cấp. Các mô hình phân cấp, mạng và quan hệ là các mô hình lôgíc do công nghệ DBMS truyền thống quản lý.

Các ngôn ngữ sẵn có trong DBMS bao gồm ngôn ngữ định nghĩa dữ liệu (DDL), ngôn ngữ thao tác dữ liệu (DML) và ngôn ngữ hỏi (QL). DDL hỗ trợ định nghĩa lược đồ cơ sở dữ liệu lôgíc. Các phép toán trên dữ liệu được xác định thì sử dụng DDL, hoặc QL. Các thao tác trên cơ sở dữ liệu bao gồm *tìm kiếm, chèn, xoá* và *cập nhật*. Để sử dụng DML, yêu cầu hiểu biết đầy đủ về mô hình, lược đồ logíc và DML được những người dùng đặc biệt sử dụng, chẳng hạn như các nhà phát triển ứng dụng. QL thì ngược lại, nó là ngôn ngữ khai báo hỗ trợ cho người dùng cuối. Ngôn ngữ DML có thể nhúng trong một ngôn ngữ lập trình thông thường, gọi là ngôn ngữ nhúng. Vì vậy, các ứng dụng sử dụng ngôn ngữ lập trình có thể đưa vào các câu lệnh của DML cho các phép toán hướng dữ liệu.

2.1 Các thành phần của DBMS

Một DBMS thông thường bao gồm nhiều môđun tương ứng với các chức năng sau:

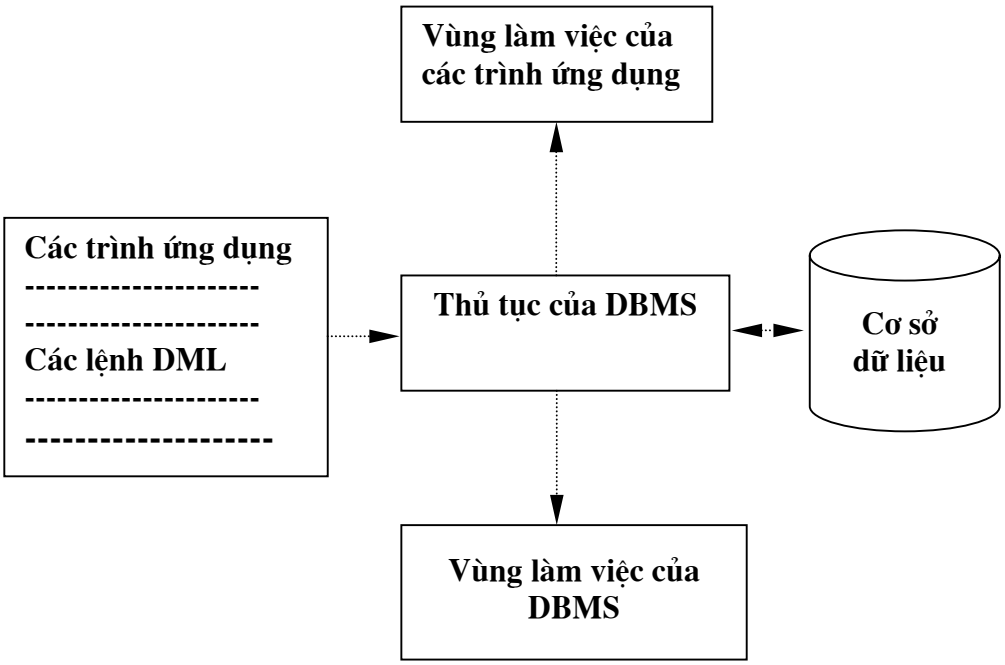
- Định nghĩa dữ liệu - DDL
- Thao tác dữ liệu - DML
- Hỏi đáp cơ sở dữ liệu - QL
- Quản trị cơ sở dữ liệu - DBMS
- Quản lý file

Tập hợp dữ liệu hỗ trợ các môđun này là:

- Các bảng mô tả cơ sở dữ liệu
- Các bảng trao quyền
- Các bảng truy nhập đồng thời

Người dùng cuối hoặc các chương trình ứng dụng có thể sử dụng dữ liệu trong cơ sở dữ liệu, thông qua các câu lệnh DML hoặc QL. Sau đó, DBMS sẽ biên dịch các câu lệnh này thông qua bộ xử lý DML và QL. Kết quả là đưa ra các câu hỏi tối ưu theo lược đồ cơ sở dữ liệu (đã được trình bày trong các bảng mô tả cơ sở dữ liệu). Những bảng này được định nghĩa thông qua các câu lệnh của DDL và được trình biên dịch DDL biên dịch. Các câu hỏi tối ưu được bộ quản trị cơ sở dữ liệu xử lý và chuyển thành các thao tác trên các file dữ liệu vật lý.

Bộ quản trị cơ sở dữ liệu cũng kiểm tra lại quyền của người dùng và các chương trình khi truy nhập dữ liệu, thông qua bảng trao quyền truy nhập. Các thao tác được phép được gửi tới bộ quản lý file. Bộ quản trị cơ sở dữ liệu cũng chịu trách nhiệm quản lý truy nhập dữ liệu đồng thời. Bộ quản trị file sẽ thực hiện các thao tác này.



Hình 1 Tương tác giữa trình ứng dụng và cơ sở dữ liệu

Hình 1 minh hoạ tương tác giữa các chương trình ứng dụng (có chứa các câu lệnh DML) và cơ sở dữ liệu. Thực hiện một câu lệnh DML tương ứng với một thủ tục của DBMS truy nhập cơ sở dữ liệu. Thủ tục lấy dữ liệu từ cơ sở dữ liệu đưa tới vùng làm việc của ứng dụng (tương ứng với câu lệnh *retrieval),* chuyển dữ liệu từ vùng làm việc vào cơ sở dữ liệu (tương ứng với các câu lệnh *insert, update*), hay xoá dữ liệu khỏi cơ sở dữ liệu (câu lệnh *delete*).

2.2 Các mức mô tả dữ liệu

DBMS mô tả dữ liệu theo nhiều mức khác nhau. Mỗi mức cung cấp một mức trừu tượng về cơ sở dữ liệu. Trong DBMS có thể có các mức mô tả sau:

Khung nhìn logíc (Logical view)

Việc xây dựng các khung nhìn tuỳ thuộc các yêu cầu của mô hình logíc và các mục đích của ứng dụng. Khung nhìn lôgíc mô tả một phần lược đồ cơ sở dữ liệu lôgíc. Nói chung, người ta thường sử dụng DDL để định nghĩa các khung nhìn lôgíc, DML để thao tác trên các khung nhìn này.

Lược đồ dữ liệu lôgíc

Ở mức này, mọi dữ liệu trong cơ sở dữ liệu được mô tả bằng mô hình lôgíc của DBMS. Các dữ liệu và quan hệ của chúng được mô tả thông qua DDL của DBMS . Các thao tác khác nhau trên lược đồ lôgíc được xác định thông qua DML của DBMS đó.

Lược đồ dữ liệu vật lý

Mức này mô tả cấu trúc lưu trữ dữ liệu trong các file trên bộ nhớ ngoài. Dữ liệu được lưu trữ dưới dạng các bản ghi (có độ dài cố định hay thay đổi) và các con trỏ trỏ tới bản ghi.

Trong mô tả dữ liệu, DBMS cho phép các mức khác nhau hỗ trợ độc lập lôgíc và độc lập vật lý. Độc lập lôgíc có nghĩa là: một lược đồ lôgíc có thể được sửa đổi mà không cần sửa đổi các chương trình ứng dụng làm việc với lược đồ này. Trong trường hợp này, mọi thay đổi trên lược đồ lôgíc cần được thay đổi lại trên các khung nhìn lôgíc có liên quan với lược đồ đó.

Độc lập vật lý có nghĩa là: một lược đồ vật lý có thể được thay đổi mà không cần phải thay đổi các ứng dụng truy nhập dữ liệu đó. Đôi khi, còn có nghĩa là: các cấu trúc lưu trữ dữ liệu vật lý có thể thay đổi mà không làm ảnh hưởng đến việc mô tả lược đồ dữ liệu lôgíc.

3. Vấn đề an toàn trong cơ sở dữ liệu

3.1 Các hiểm hoạ đối với an toàn cơ sở dữ liệu

Một hiểm hoạ có thể được xác định khi đối phương (người, hoặc nhóm người) sử dụng các kỹ thuật đặc biệt để tiếp cận nhằm khám phá, sửa đổi trái phép thông tin quan trọng do hệ thống quản lý.

Các xâm phạm tính an toàn cơ sở dữ liệu bao gồm *đọc, sửa*, *xoá dữ liệu trái phép*. Thông qua những xâm phạm này, đối phương có thể:

- Khai thác dữ liệu trái phép thông qua suy diễn thông tin được phép.
- Sửa đổi dữ liệu trái phép.
- Từ chối dịch vụ hợp pháp.

Các hiểm hoạ an toàn có thể được phân lớp, tuỳ theo cách thức xuất hiện của chúng, là hiểm hoạ *có chủ ý* và *vô ý (ngẫu nhiên)*.

Hiểm hoạ ngẫu nhiên là các hiểm hoạ thông thường độc lập với các điều khiển gây phá hỏng cơ sở dữ liệu, chúng thường liên quan tới các trường hợp sau:

- Các thảm hoạ trong thiên nhiên, chẳng hạn như động đất, hoả hoạn, lụt lội... có thể phá hỏng các hệ thống phần cứng, hệ thống lưu giữ số liệu, dẫn đến các xâm phạm tính toàn vẹn và sẵn sàng của hệ thống.
- Các lỗi phần cứng hay phần mềm có thể dẫn đến việc áp dụng các chính sách an toàn không đúng, từ đó cho phép truy nhập, đọc, sửa đổi dữ liệu trái phép, hoặc từ chối dịch vụ đối với người dùng hợp pháp.
- Các sai phạm vô ý do con người gây ra, chẳng hạn như nhập dữ liệu đầu vào không chính xác, hay sử dụng các ứng dụng không đúng, hậu quả cũng tương tự như các nguyên nhân do lỗi phần mềm hay lỗi kỹ thuật gây ra.

Những xâm phạm trên liên quan đến hai lớp người dùng sau:

- Người dùng được phép là người có thể lạm dụng quyền, sử dụng vượt quá quyền hạn được phép của họ.
- Đối phương là người, hay nhóm người truy nhập thông tin trái phép, có thể là những người nằm ngoài tổ chức hay bên trong tổ chức. Họ tiến hành các hành vi phá hoại phần mềm cơ sở dữ liệu hay phần cứng của hệ thống, hoặc đọc ghi dữ liệu trái phép. Trong cả hai trường hợp trên, họ đều thực hiện với chủ ý rõ ràng.

3.2 Các yêu cầu bảo vệ cơ sở dữ liệu

Bảo vệ cơ sở dữ liệu khỏi các hiểm hoạ, có nghĩa là bảo vệ tài nguyên, đặc biệt là dữ liệu khỏi các thảm hoạ, hoặc truy nhập trái phép. Các yêu cầu bảo vệ cơ sở dữ liệu gồm:

Bảo vệ chống truy nhập trái phép

Đây là một vấn đề cơ bản, bao gồm trao quyền truy nhập cơ sở dữ liệu cho người dùng hợp pháp. Yêu cầu truy nhập của ứng dụng, hoặc người dùng phải được DBMS kiểm tra. Kiểm soát truy nhập cơ sở dữ liệu phức tạp hơn kiểm soát truy

nhập file. Việc kiểm soát cần tiến hành trên các đối tượng dữ liệu ở mức thấp hơn mức file (chẳng hạn như các bản ghi, các thuộc tính và các giá trị). Dữ liệu trong cơ sở dữ liệu thường có quan hệ với nhau về ngữ nghĩa, do đó cho phép người sử dụng có thể biết được giá trị của dữ liệu mà không cần truy nhập trực tiếp, bằng cách suy diễn từ các giá trị đã biết.

Bảo vệ chống suy diễn

Suy diễn là khả năng có được các thông tin bí mật từ những thông tin không bí mật. Đặc biệt, suy diễn ảnh hưởng tới các cơ sở dữ liệu thống kê, trong đó người dùng không được phép dò xét thông tin của các cá thể khác từ các dữ liệu thống kê đó.

Bảo vệ toàn vẹn cơ sở dữ liệu

Yêu cầu này bảo vệ cơ sở dữ liệu khỏi các truy nhập trái phép mà có thể dẫn đến việc thay đổi nội dung dữ liệu. Các lỗi, virus, hỏng hóc trong hệ thống có thể gây hỏng dữ liệu. DBMS đưa ra dạng bảo vệ này, thông qua các kiểm soát về sự đúng đắn của hệ thống, các thủ tục sao lưu, phục hồi và các thủ tục an toàn đặc biệt.

Để duy trì tính tương thích của cơ sở dữ liệu, mỗi giao tác phải là một đơn vị tính toán tin cậy và tương thích.

Hệ thống khôi phục (*recovery system*) sử dụng nhật ký. Với mỗi giao tác, nhật ký ghi lại các phép toán đã được thực hiện trên dữ liệu (chẳng hạn như *read, write, delete, insert*), cũng như các phép toán điều khiển giao tác (chẳng hạn như *commit, abort*), cả giá trị cũ và mới của các bản ghi kéo theo. Hệ thống phục hồi đọc file nhật ký để xác định giáo tác nào bị huỷ bỏ và giao tác nào cần phải thực hiện lại. Huỷ một giao tác có nghĩa là phục hồi lại giá trị cũ của mỗi phép toán trên bản ghi kéo theo. Thực hiện lại giao tác có nghĩa là cập nhật giá trị mới của mỗi phép toán vào bản ghi kéo theo.

Các thủ tục an toàn đặc biệt bảo vệ dữ liệu không bị truy nhập trái phép. Xây dựng mô hình, thiết kế và thực hiện các thủ tục này là một trong các mục tiêu an toàn cơ sở dữ liệu.

Toàn vẹn dữ liệu thao tác

Yêu cầu này đảm bảo tính tương thích lôgíc của dữ liệu khi có nhiều giao tác thực hiện đồng thời.

Bộ quản lý tương tranh trong DBMS đảm bảo tính chất khả tuần tự và cô lập của các giao tác. Khả tuần tự có nghĩa là kết quả của việc thực hiện đồng thời một tập hợp các giao tác giống với việc thực hiện tuần tự các giao tác này. Tính cô lập để chỉ sự độc lập giữa các giao tác, tránh được hiệu ứng Domino, trong đó việc huỷ bỏ một giao tác dẫn đến việc huỷ bỏ các giao tác khác (theo kiểu thác đổ).

Vấn đề đảm bảo truy nhập đồng thời vào cùng một thực thể dữ liệu, từ các giao tác khác nhau, nhưng không làm ảnh hưởng đến tính tương thích của dữ liệu, được giải quyết bằng các kỹ thuật khoá.

Các kỹ thuật khoá và giải phóng khoá được thực hiện theo nguyên tắc: khoá các mục dữ liệu trong một khoảng thời gian cần thiết để thực hiện phép toán và giải phóng khoá khi phép toán đã hoàn tất. Tuy nhiên kỹ thuật này không đảm bảo tính khả tuần tự. Nhược điểm này được khắc phục bằng cách sử dụng kỹ thuật khoá hai pha.

Toàn vẹn ngữ nghĩa của dữ liệu

Yêu cầu này đảm bảo tính tương thích lôgíc của các dữ liệu bị thay đổi, bằng cách kiểm tra các giá trị dữ liệu có nằm trong khoảng cho phép hay không. Các hạn chế (trên các giá trị dữ liệu) được biểu diễn như là các ràng buộc toàn vẹn. Các ràng buộc có thể được xác định trên toàn bộ cơ sở dữ liệu hoặc là cho một số các giao tác.

Khả năng lưu vết và kiểm tra

Yêu cầu này bao gồm khả năng ghi lại mọi truy nhập tới dữ liệu (với các phép toán *read* và *write)*. Khả năng kiểm tra và lưu vết đảm bảo tính toàn vẹn dữ liệu vật lý và trợ giúp cho việc phân tích dãy truy nhập vào cơ sở dữ liệu.

Xác thực người dùng

Yêu cầu này thực sự cần thiết để xác định tính duy nhất của người dùng. Định danh người dùng làm cơ sở cho việc trao quyền. Người dùng được phép truy nhập dữ liệu, khi hệ thống xác định được người dùng này là hợp pháp.

Quản lý và bảo vệ dữ liệu nhạy cảm

Có những cơ sở dữ liệu chứa nhiều dữ liệu nhạy cảm (là những dữ liệu không nên đưa ra công bố công khai). Có những cơ sở dữ liệu chỉ chứa các dữ liệu nhạy cảm, chẳng hạn như dữ liệu quân sự, còn có các cơ sở dữ liệu mang tính công cộng, chẳng hạn như các cơ sở dữ liệu của thư viện.

Các cơ sở dữ liệu bao gồm cả thông tin nhạy cảm và thông tin thường cần phải có các chính sách quản lý phức tạp hơn. Một mục dữ liệu là nhạy cảm khi chúng được người quản trị cơ sở dữ liệu (DBA) khai báo là nhạy cảm.

Kiểm soát truy nhập vào các cơ sở dữ liệu bao hàm: bảo vệ tính tin cậy của dữ liệu nhậy cảm và chỉ cho phép người dùng hợp pháp truy nhập vào. Những người dùng này được trao một số quyền thao tác nào đó trên dữ liệu và không được phép lan truyền chúng. Do vậy, người dùng có thể truy nhập vào các tập con dữ liệu nhạy cảm.

Bảo vệ nhiều mức

Bảo vệ nhiều mức bao gồm một tập hợp các yêu cầu bảo vệ. Thông tin có thể được phân loại thành nhiều mức khác nhau, ví dụ các cơ sở dữ liệu quân sự cần được phân loại chi tiết hơn (mịn hơn) các cơ sở dữ liệu thông thường, có thể có nhiều mức nhạy cảm khác nhau. Mục đích của bảo vệ nhiều mức là phân loại các mục thông tin khác nhau, đồng thời phân quyền cho các mức truy nhập khác nhau vào các mục riêng biệt. Một yêu cầu nữa đối với bảo vệ nhiều mức là khả năng gán mức cho các thông tin.

Sự hạn chế

Mục đích của việc hạn chế là tránh chuyển các thông tin không mong muốn giữa các chương trình trong hệ thống, ví dụ chuyển dữ liệu quan trọng tới các chương trình không có thẩm quyền. Các kênh được phép cung cấp thông tin thông qua các hoạt động được phép, như soạn thảo hay biên dịch một file. Kênh bộ nhớ là các vùng bộ nhớ, nơi một chương trình có thể lưu giữ dữ liệu, các chương trình khác cũng có thể đọc dữ liệu này. Kênh ngầm là kênh truyền thông dựa trên việc sử dụng tài nguyên mà không có ý định truyền thông giữa các tiến trình của hệ thống.

4. Kiểm soát an toàn

Có thể bảo vệ được cơ sở dữ liệu thông qua các phương pháp an toàn sau:

- Kiểm soát luồng
- Kiểm soát suy diễn
- Kiểm soát truy nhập

Với các kiểm soát này, kỹ thuật mật mã có thể được đưa vào để mã hoá dữ liệu với khoá mã bí mật. Thông qua kỹ thuật này, bí mật của thông tin được bảo đảm, bằng cách tạo ra dữ liệu mà ai cũng có thể nhìn được nhưng chỉ người dùng hợp pháp mới hiểu được .

4.1 Kiểm soát luồng

Các kiểm soát luồng điều chỉnh phân bố luồng thông tin giữa các đối tượng có khả năng truy nhập. Một luồng giữa đối tượng X và đối tượng Y xuất hiện khi có một lệnh đọc (*read*) giá trị từ X và ghi (*write*) giá trị vào Y. Kiểm soát luồng là kiểm tra xem thông tin có trong một số đối tượng có chảy vào các đối tượng có mức bảo vệ thấp hơn hay không.

Các chính sách kiểm soát luồng cần phải chỉ ra các luồng có thể được chấp nhận, hoặc phải điều chỉnh.

Thông thường, trong kiểm soát luồng người ta phải tính đến việc phân loại các phần tử của hệ thống, đó là các *đối tượng* và *chủ thể*. Các phép toán được phép (*read* và *write*) dựa trên quan hệ giữa các lớp. Phép toán *read* đối tượng có mức bảo vệ cao hơn bị kiểm soát nhiều hơn. Kiểm soát luồng ngặn chặn việc chuyển thông tin vào các mức dễ truy nhập hơn. Vấn đề của chính sách kiểm soát luồng được giải quyết bằng cách xác định các phép toán cho phép chuyển thông tin tới các mức thấp hơn mà vẫn giữ nguyên được mức nhạy cảm của những đối tượng này.

4.2 Kiểm soát suy diễn

Kiểm soát suy diễn nhằm mục đích bảo vệ dữ liệu không bị khám phá gián tiếp. Kênh suy diễn là kênh mà ở đó người dùng có thể tìm thấy mục dữ liệu X, sau đó sử dụng X để suy ra mục dữ liệu Y, thông qua Y=f(X).

Các kênh suy diễn chính trong hệ thống là:

(1) *Truy nhập gián tiếp*: điều này xảy ra khi người (không được trao quyền) khám phá ra bộ dữ liệu Y thông qua các câu hỏi truy vấn được phép trên dữ liệu X, cùng với các điều kiện trên Y.

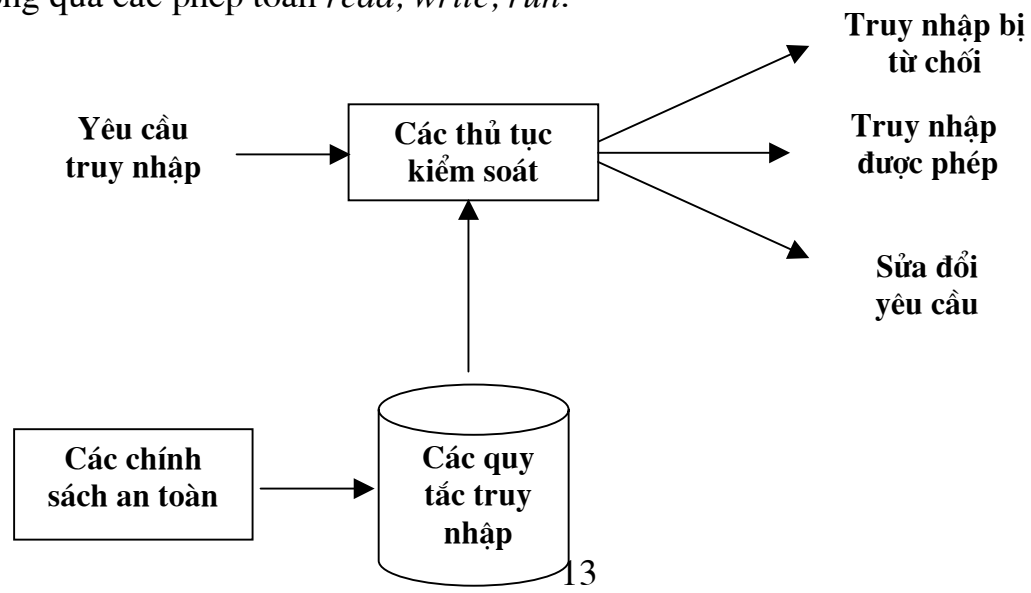
(2) Dữ liệu tương quan: Dữ liệu tương quan là một kênh suy diễn đặc trưng, xảy ra khi dữ liệu có thể nhìn thấy được X và dữ liệu không thể nhìn thấy được Y kết nối với nhau mặt ngữ nghĩa. Kết quả là có thể khám phá được thông tin về Y nhờ đọc X.

(3) Thiếu dữ liệu: Kênh thiếu dữ liệu là một kênh suy diễn mà qua đó, người dùng có thể biết được sự tồn tại của một tập giá trị X. Đặc biệt, người dùng có thể tìm được tên của đối tượng, mặc dù họ không được phép truy nhập vào thông tin chứa trong đó.

Suy diễn thống kê là một khía cạnh khác của suy diễn dữ liệu. Trong các cơ sở dữ liệu thống kê, người dùng không được phép truy nhập vào các dữ liệu đơn lẻ, chỉ được phép truy nhập vào dữ liệu thông qua các hàm thống kê. Tuy nhiên với một người có kinh nghiệm, anh ta vẫn có thể khám phá được dữ liệu thông qua các thống kê đó.

4.3 Kiểm soát truy nhập

Kiểm soát truy nhập trong các hệ thống thông tin là đảm bảo mọi truy nhập trực tiếp vào các đối tượng của hệ thống tuân theo các kiểu và các quy tắc đã được xác định trong chính sách bảo vệ. Một hệ thống kiểm soát truy nhập (hình 2) bao gồm các chủ thể (người dùng, tiến trình) truy nhập vào đối tượng (dữ liệu, chương trình) thông qua các phép toán *read, write, run*.



Hình 2 Hệ thống kiểm soát truy nhập

Xét về mặt chức năng, nó bao gồm hai thành phần:

1) Tập các chính sách và quy tắc truy nhập: bao gồm các thông tin về chế độ truy nhập mà các chủ thể có thể có được khi truy nhập các đối tượng.

2) Tập các thủ tục kiểm soát (các kỹ thuật an toàn): Kiểm tra các câu hỏi (các yêu cầu truy nhập) dựa vào các quy tắc đã được xác định (quá trình phê chuẩn câu hỏi); các câu hỏi này có thể được phép, bị từ chối hoặc bị sửa đổi.

Các chính sách an toàn

Chính sách an toàn của hệ thống là các hướng dẫn ở mức cao, có liên quan đến việc thiết kế và quản lý hệ thống trao quyền. Nhìn chung, chúng biểu diễn các lựa chọn cơ bản nhằm đảm bảo mục tiêu an toàn dữ liệu. Chính sách an toàn định nghĩa các nguyên tắc, trong đó quy định truy nhập nào được trao hoặc bị từ chối.

Các quy tắc trao quyền (quy tắc truy nhập) là các biểu diễn của chính sách an toàn; Chúng quyết định hành vi của hệ thống trong thời gian chạy. Các chính sách an toàn nên xác định: làm thế nào để quản lý được tập các quy tắc quyền (chèn và sửa đổi). Sau đây là một ví dụ về chính sách an toàn.

Trong vấn đề giới hạn truy nhập, một câu hỏi đặt ra là "Mỗi chủ thể có được phép truy nhập bao nhiêu thông tin". Chúng ta có hai chính sách sau đây:

1) *Chính sách đặc quyền tối tiểu*: còn được gọi là chính sách "cần - để - biết" (*need-to-know*). Theo chính sách này, các chủ thể của hệ thống nên sử dụng một lượng thông tin tối thiểu cần cho hoạt động của chúng. Đôi khi, việc ước tính lượng thông tin tối thiểu này là rất khó. Điểm hạn chế của chính sách này đưa ra các hạn chế khá lớn và vô ích đối với các chủ thể vô hại.

2) *Chính sách đặc quyền tối đa*: dựa vào nguyên tắc "khả năng sẵn sàng tối đa" của dữ liệu, vì vậy mức độ chia xẻ là cực đại. Chính sách này phù hợp với các môi trường (chẳng hạn như trường đại học, trung tâm nghiên cứu), việc bảo vệ nghiêm ngặt tại những nơi này thực sự không cần thiết, do các yêu cầu về độ tin cậy người dùng và trao đổi dữ liệu.

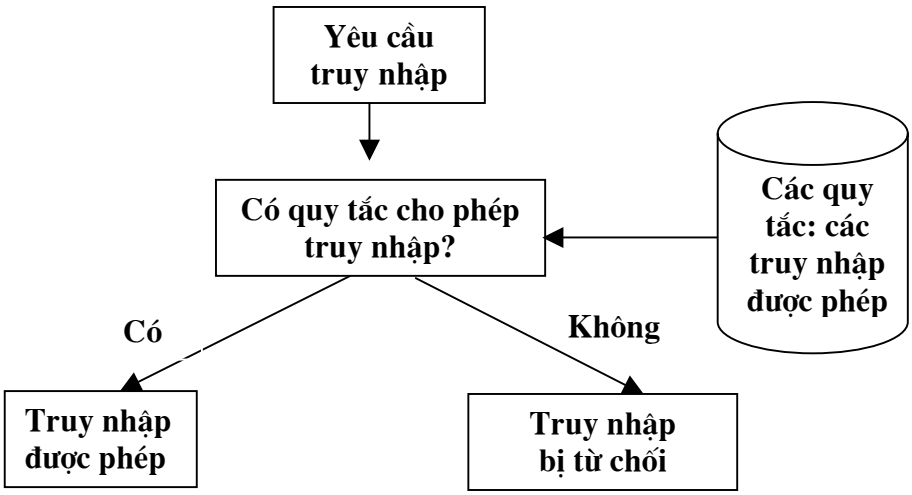
Trong một hệ thống khép kín, chỉ cho phép các truy nhập được phép. Trong một hệ thống mở, cho phép các truy nhập không bị cấm.

Chính sách của một hệ thống khép kín chỉ rõ, với mỗi chủ thể: các quy tắc trao quyền hiện có xác định các đặc quyền truy nhập mà chủ thể đó có được trên các đối tượng của hệ thống. Đây là những quyền mà chủ thể được trao, thông qua cơ chế kiểm soát. Chính sách của một hệ thống mở chỉ rõ, đối với mỗi chủ thể: các quy tắc trao quyền hiện có xác định các đặc quyền mà chủ thể không nắm giữ trên các đối tượng của hệ thống. Đây là những quyền mà chủ thể bị từ chối, thông qua cơ chế kiểm soát.

Khi việc quyết định dựa vào các chiến lược an toàn, sự lựa chọn phụ thuộc vào các đặc điểm và yêu cầu của môi trường, người dùng, ứng dụng,.v.v. Một hệ thống khép kín tuân theo chính sách đặc quyền tối thiểu, trong khi đó hệ thống mở tuân theo chính sách đặc quyền tối đa. Việc bảo vệ trong các hệ thống khép kín cao hơn. Các lỗi (chẳng hạn như một quy tắc thiếu) có thể từ chối truy nhập được phép, nhưng điều này không gây thiệt hại, ngược lại trong các hệ thống mở, điều này có thể dẫn đến việc trao các truy nhập trái phép.

Các hệ thống khép kín cho phép đánh giá tình trạng trao quyền dễ dàng hơn, vì các đặc quyền do người dùng nắm giữ, chính vì vậy, kiểu hệ thống này thường được lựa chọn nhiều hơn. Tuy nhiên, việc chọn lựa cũng phụ thuộc vào dạng môi trường và các yêu cầu bảo vệ.

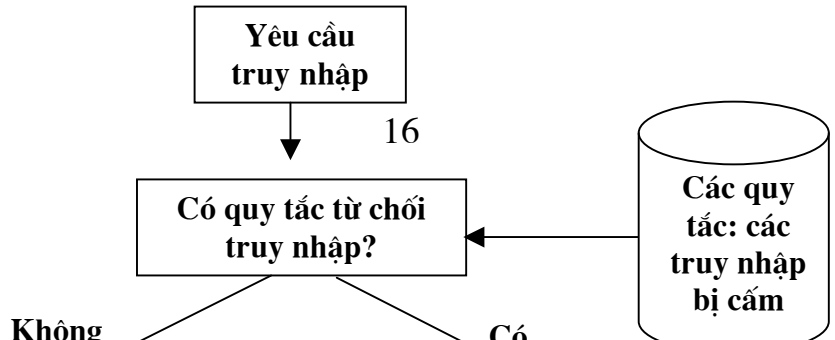
Các kiểm soát truy nhập (tùy thuộc vào chính sách của các hệ thống khép kín và mở) được minh hoạ trong hình 3 và 4.



Hình 3 Kiểm soát truy nhập trong các hệ thống khép kín

Trong một hệ thống an toàn, việc định nghĩa các chính sách quản lý quyền là xác định "ai" có thể trao quyền hoặc huỷ bỏ quyền truy nhập.

Việc trao và huỷ bỏ không phải lúc nào cũng thuộc quyền của người trao quyền hoặc nhân viên an ninh. Đôi khi, việc quản lý trao quyền đòi hỏi sự tham gia của nhiều người khác nhau. Đây là một đặc thù của hệ thống phân tán, trong đó các hệ thống cục bộ khác nhau thường được quản lý tự trị. Điều này cũng xảy ra trong các hệ thông tin lớn, cơ sở dữ liệu được phân hoạch lôgíc thành các cơ sở dữ liệu khác nhau, mỗi phần được một DBA địa phương quản lý.



Hình 4 Kiểm soát truy nhập trong các hệ thống mở

Sự lựa chọn giữa quản lý tập trung và phi tập trung là một chính sách an toàn. Tuy nhiên, có thể có các chính sách trung gian, ví dụ:

- *Trao quyền phi tập trung phân cấp*: trong đó, người trao quyền trung tâm có trách nhiệm chia nhỏ trách nhiệm quản trị cơ sở dữ liệu cho những người quản trị cấp dưới. Ví dụ, người trao quyền trung tâm có thể chỉ định hoặc không sử dụng người quản trị cấp dưới của anh ta.

- *Quyền sở hữu* : người tạo ra đối tượng (ví dụ, một bảng trong cơ sở dữ liệu quan hệ) là người sở hữu đối tượng đó (điều này là mặc định). Do vậy, anh ta có quyền trao hoặc huỷ bỏ truy nhập tới đối tượng đó, đôi khi cần có sự đồng ý của người quản trị trung tâm.

- *Quyền hợp tác*: Việc trao các quyền đặc biệt trên một số tài nguyên nào đó không thể chỉ do một người quyết định mà phải có sự đồng ý của một nhóm người dùng xác định.

Các chính sách kiểm soát truy nhập: xác định cách thức nhóm các chủ thể và các đối tượng của hệ thống để chia xẻ các chế độ truy nhập tuỳ thuộc vào các quyền và các quy tắc định trước. Hơn nữa, chính sách xác định các quyền truy nhập có thể được chuyển và chuyển như thế nào.

Những người dùng (trong cùng một nhóm, hoặc cùng mức phân loại) có một số đặc quyền, hoặc tài nguyên (có các yêu cầu bảo vệ chung) đơn giản hoá việc đặc tả các chính sách an toàn và việc thực thi các cơ chế an toàn. Vì vậy, người ta đã đề xuất nhiều tiêu chuẩn nhóm khác nhau, chẳng hạn như:

- Mức thiết kế: phân hoạch người dùng.

- Mức thực thi: cách thức quản lý việc chuyển người dùng giữa các mức khác nhau.

Các kiểm soát truy nhập được ánh xạ vào các kiểm soát luồng thông tin giữa các mức khác nhau. Thủ tục này được sử dụng rộng rãi trong các hệ thống an toàn đa mức trong quân sự, trong đó các chính sách kiểm soát truy nhập thực chất là các chính sách kiểm soát luồng thông tin.

Các hệ thống đa mức đã thành công, do chúng được xây dựng trên các mô hình an toàn được nghiên cứu đầy đủ về mặt lý thuyết.

Kiểm soát truy nhập bắt buộc (MAC) hạn chế truy nhập của các chủ thể vào các đối tượng, bằng cách sử dụng các nhãn an toàn. Kiểm soát truy nhập tuỳ ý (DAC) cho phép lan truyền các quyền truy nhập từ chủ thể này đến chủ thể khác.

Chính sách bắt buộc trong kiểm soát truy nhập được áp dụng cho các thông tin có yêu cầu bảo vệ nghiêm ngặt, trong các môi trường mà ở đó dữ liệu hệ thống có thể được phân loại và người dùng được xác định rõ ràng. Chính sách bắt buộc cũng có thể được định nghĩa như là một chính sách kiểm soát luồng, bởi vì nó ngăn chặn dòng thông tin chảy vào các đối tượng có mức phân loại thấp hơn. Chính sách bắt buộc quyết định truy nhập vào dữ liệu, thông qua việc định nghĩa các lớp an toàn của chủ thể và đối tượng. Hai đặc điểm chính của lớp đối tượng an toàn là: mức phân loại phản ánh thông tin có trong đó và loại (vùng ứng dụng) mà thông tin đối tượng đề cập đến. Ví dụ, các mức phân loại như sau:

0= Thông thường

1= Mật

2= Tuyệt mật

3= Tối mật

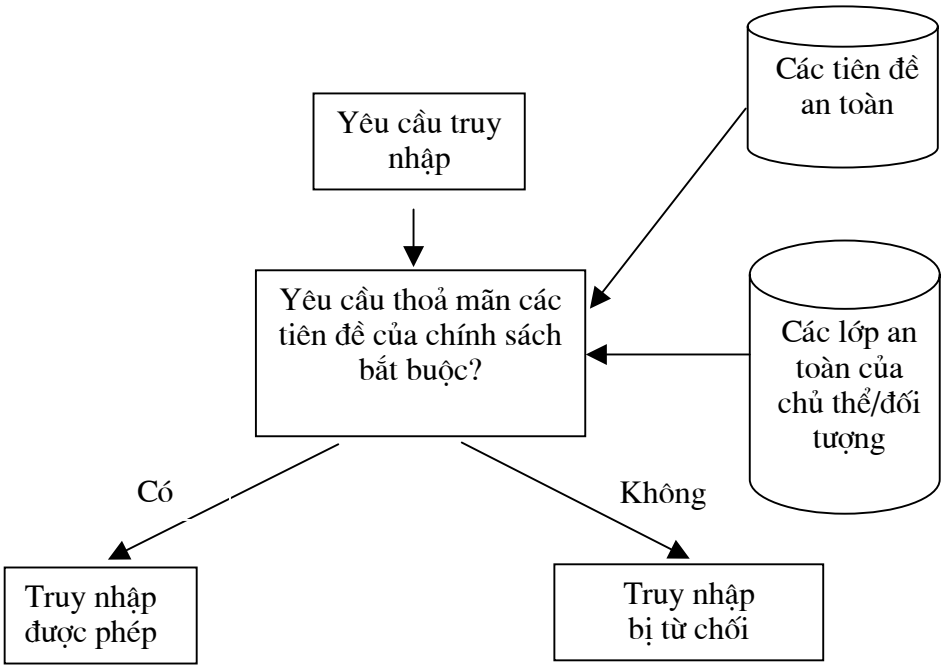
Loại phản ánh các vùng của hệ thống, hoặc các bộ phận của tổ chức. Với *m* vùng hệ thống, có thể chia tối đa thành 2^m loại.

Mỗi chủ thể và đối tượng được gán một lớp an toàn, bao gồm một mức nhạy cảm và một tập hợp các loại. Phân loại các chủ thể phản ánh mức độ tin cậy có thể được

gán cho chủ thể đó và vùng ứng dụng mà nó làm việc. Phân loại đối tượng phản ánh mức độ nhạy cảm của thông tin có trong đối tượng.

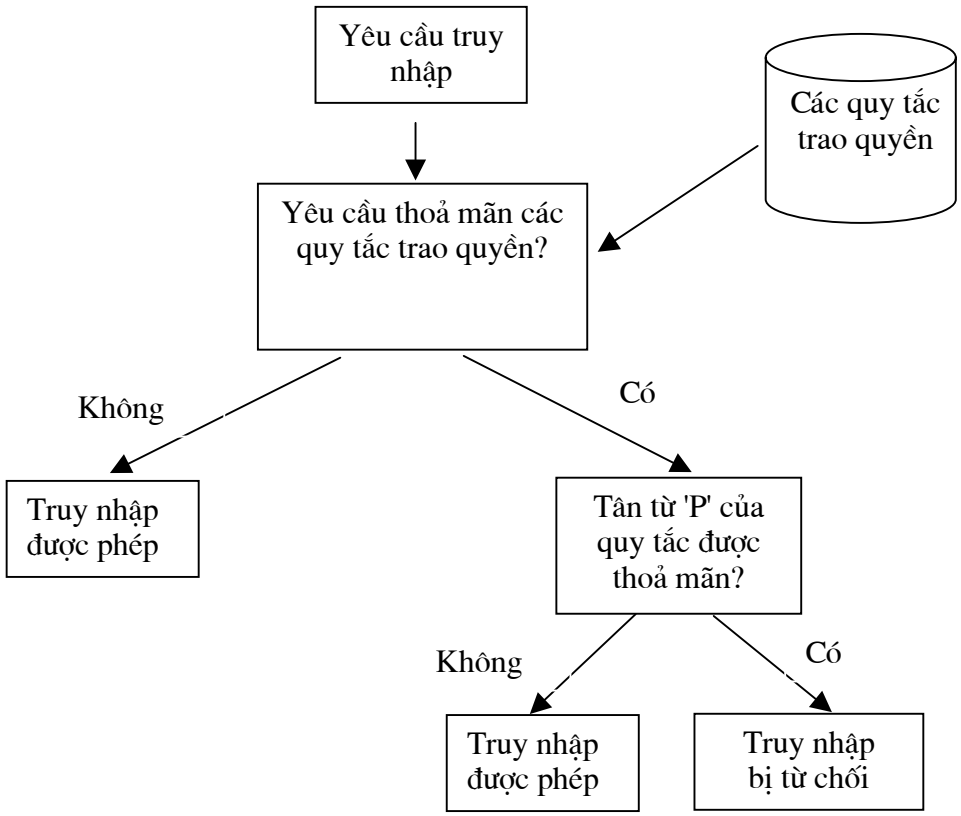
Một tập hợp các tiên đề xác định các quan hệ được kiểm tra giữa lớp chủ thể và lớp đối tượng, cho phép các chủ thể truy nhập vào các đối tượng theo tiêu chuẩn an toàn. Những quan hệ này phụ thuộc vào chế độ truy nhập.

Về việc chuyển giao quyền truy nhập, không thể thay đổi các quyền đã được gán, mọi thay đổi chỉ được phép khi có sự đồng ý của người trao quyền. Điều này có nghĩa là, người trao quyền kiểm soát toàn bộ hệ thống trao quyền. Kiểm soát truy nhập thông qua các chính sách bắt buộc được minh hoạ trong hình 5.



Hình 5 Kiểm soát truy nhập bắt buộc

Chính sách tùy ý chỉ rõ những đặc quyền mà mỗi chủ thể có thể có được trên các đối tượng của hệ thống. Các yêu cầu truy nhập được kiểm tra, thông qua một cơ chế kiểm soát tuỳ ý, truy nhập chỉ được trao cho các chủ thể thoả mãn các quy tắc trao quyền hiện có (hình 6).



Hình 6 Kiểm soát truy nhập tuỳ ý

Chính sách tuỳ ý dựa vào định danh của người dùng có yêu cầu truy nhập. Điều này ngầm định rằng, việc phân quyền kiểm soát dựa vào quyền sở hữu. Tuy nhiên, chính sách tuỳ ý cũng phù hợp với quản trị tập trung. Trong trường hợp này, quyền được người quản trị hệ thống quản lý: quản trị phi tập trung ý muốn nói đến các chính sách kiểm soát tuỳ ý. Chính sách tuỳ ý cần các cơ chế trao quyền phức tạp hơn, nhằm tránh mất quyền kiểm soát khi lan truyền quyền từ người trao quyền, hoặc những người có trách nhiệm khác.

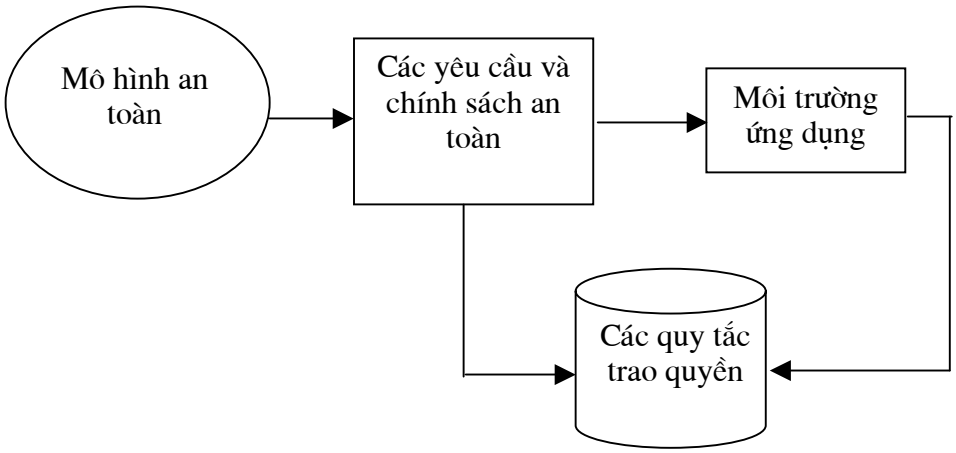
Sự thu hồi quyền đã được lan truyền là một vấn đề khác. Với mỗi quyền bị thu hồi, người dùng (người đã được trao hoặc nhận quyền đó) phải được hệ thống nhận dạng (xác định). Hiện tồn tại nhiều chính sách thu hồi khác nhau cho mục đích này. DAC có nhược điểm sau: nó cho phép đọc thông tin từ một đối tượng và chuyển đến một đối tượng khác mà có thể được ghi bởi chủ thể;

Các chính sách bắt buộc và tuỳ ý là không loại trừ lẫn nhau. Chúng có thể được kết hợp với nhau: chính sách bắt buộc được áp dụng cho kiểm soát trao quyền, trong khi đó chính sách tuỳ ý được áp dụng cho kiểm soát truy nhập.

Các quy tắc trao quyền

Như đã trình bày ở trên, nhiệm vụ của người cấp quyền là chuyển đổi các yêu cầu và các chính sách an toàn thành các quy tắc trao quyền. Thông thường, tổ chức xác định các yêu cầu an toàn và người dùng nhận biết chúng thông qua kinh nghiệm của họ.

Các quy tắc trao quyền được biểu diễn đúng với môi trường phần mềm/phần cứng của hệ thống bảo vệ và các chính sách an toàn được chấp thuận. Quá trình thiết kế một hệ thống an toàn phải đưa ra được một mô hình và mô hình này hỗ trợ người trao quyền khi ánh xạ các yêu cầu vào các quy tắc, tuỳ theo các chính sách an toàn cần quan tâm (Hình 7).



Hình 7 Thiết kế các quy tắc trao quyền

Một ví dụ về ma trận quyền được trình bày trong bảng 1, trong đó *R= Read, W= Write, EXC= Execute, CR= Create*, và *DEL= Delete*. Đây là một trường hợp kiểm soát truy nhập đơn giản dựa vào tên đối tượng, được gọi là truy nhập phụ thuộc tên (*name-dependent access*). Các quyền có thể bao gồm nhiều quy tắc an toàn phức tạp hơn, chúng xác định các ràng buộc truy nhập giữa chủ thể và đối tượng.

Một tân từ (*predicate*) cũng có thể được xem là biểu thức của một số biến hệ thống, chẳng hạn như ngày, giờ và nguồn truy vấn, vì vậy đã thiết lập cơ sở cho kiểm soát phụ thuộc ngữ cảnh (*context- dependent control*). Một ví dụ về kiểm soát phụ thuộc ngữ cảnh là không thể truy nhập vào thông tin được phân loại thông qua một đăng nhập từ xa (*remote login*), hoặc chỉ cập nhật thông tin về lương vào thời điểm cuối của năm. Kiểm soát phụ thuộc nội dung (*content-dependent control*) nằm ngoài phạm vi của các hệ điều hành, do DBMS cung cấp. Với kiểm soát phụ thuộc ngữ cảnh (*context-dependent control*), một phần do hệ điều hành cung cấp, một phần do DBMS cung cấp.

Bảng 1 Ma trận quyền

	Đối tượng		
Chủ thể	**File F1**	**File F2**	**File F3**
Người dùng 1	R,W	EXEC	EXEC
Người dùng 2	-	-	CR, DEL
Chương trình P1	R,W	R	-

Các quy tắc an toàn cũng nên xác định các kết hợp dữ liệu không được phép, cần phải xem độ nhạy cảm của dữ liệu có tăng lên sau khi kết hợp hay không. Ví dụ, người dùng có thể được phép đọc tên và các giá trị lương của nhân viên một cách riêng lẻ, nhưng không được phép đọc kết hợp "tên - lương", nếu không họ có thể liên hệ lương với từng nhân viên cụ thể.

Việc truy nhập vào các chương trình xử lý dữ liệu (ví dụ, một số chương trình hệ thống hoặc ứng dụng) cũng cần được kiểm soát. Các vấn đề và cơ chế liên quan đến việc bảo vệ dữ liệu cần được mở rộng, nhằm giải quyết các vấn đề phức tạp hơn về bảo vệ logíc tất cả tài nguyên hệ thống. Với các chính sách tuỳ ý, việc trao hoặc huỷ bỏ quyền truy nhập phụ thuộc vào một vài người trao quyền, một quy tắc an toàn có thể là bộ 6 *{ a, s, o, t, p ,f}*, trong đó *a* là người trao quyền, *s* là chủ thể được trao quyền *{o, t, p}*, *f* là cờ sao chép (*copy flag*) mô tả khả năng *s* chuyển quyền *{o, t, p}* cho các chủ thể khác. Các chính sách an toàn (liên quan đến việc chuyển quyền truy nhập) quyết định sự xuất hiện của cờ này, cũng như việc sử dụng nó. Ví dụ trong một vài hệ thống, cờ được xác lập lại sau *n* lần trao, vì vậy cho phép chuyển quyền sâu *n* mức.

Cơ chế an toàn

Hệ thống kiểm soát truy nhập dựa vào các cơ chế an toàn là các chức năng thực hiện các quy tắc và chính sách an toàn. Các cơ chế an toàn liên quan đến việc ngăn chặn truy nhập trái phép (các cơ chế kiểm soát truy nhập) và phát hiện truy nhập trái phép (cơ chế phát hiện xâm nhập và kiểm toán). Muốn ngăn chặn và phát hiện tốt đòi hỏi các cơ chế xác thực tốt. Các cơ chế kiểm soát truy nhập được chọn lựa nhiều hơn. Đôi khi, phát hiện là một tuỳ chọn, ví dụ khả năng giải trình việc sử

dụng đúng đắn các đặc quyền, hoặc chống lại việc sửa đổi các thông báo trên mạng.

Các cơ chế an toàn có thể được thực thi thông qua phần cứng, phần mềm hoặc thông qua các thủ tục quản lý. Khi phát triển hệ thống an toàn, các chính sách và cơ chế nên được tách rời để có thể:

- Thảo luận (một cách độc lập) các quy tắc truy nhập về các cơ chế thực hiện. Điều này cho phép các nhà thiết kế tập trung vào tính đúng đắn của các yêu cầu an toàn, cũng như tính tương thích của các chính sách an toàn.
- So sánh các chính sách kiểm soát truy nhập khác nhau, hoặc các cơ chế thực hiện khác nhau cho cùng một chính sách.
- Thiết kế các cơ chế có khả năng thực hiện các chính sách khác nhau. Điều này cần thiết khi các chính sách cần thay đổi động, tuỳ thuộc vào sự thay đổi của môi trường ứng dụng và các yêu cầu bảo vệ. Chính sách an toàn nên có quan hệ chặt chẽ với cơ chế thực hiện. Mọi thay đổi của chính sách phải phù hợp với hệ thống kiểm soát.

Để có được các cơ chế tuân theo các chính sách (đã được thiết kế) là một vấn đề mang tính quyết định. Trong thực tế, việc thực hiện không đúng một chính sách an toàn dẫn đến các quy tắc truy nhập không đúng, hoặc hỗ trợ không đầy đủ chính sách bảo vệ. Hai kiểu lỗi hệ thống cơ bản có thể xuất phát từ việc thực thi không đúng:

(1) Từ chối truy nhập được phép

(2) Cho phép truy nhập đã bị cấm

Các cơ chế bên ngoài

Chúng bao gồm các biện pháp kiểm soát vật lý và quản lý, có thể ngăn ngừa truy nhập trái phép vào tài nguyên vật lý (phòng, thiết bị đầu cuối, các thiết bị khác), vì vậy chỉ cho phép các truy nhập được phép. Ngoài ra còn có các thiết bị có khả năng bảo vệ chống lại các hiểm hoạ. Tuy nhiên, để có được bảo vệ đầy đủ là không thể, đặc biệt trong các môi trường có tấn công hoặc xâm phạm ngẫu nhiên. Mục tiêu là giảm đến mức tối thiểu các thiệt hại. Điều này có nghĩa là:

- Giảm đến mức tối thiểu các xâm phạm;
- Giảm đến mức tối thiểu các thiệt hại;
- Cung cấp các thủ tục khôi phục.

Các cơ chế bên trong

Các biện pháp bảo vệ được áp dụng khi người dùng bỏ qua, hoặc nhận được quyền truy nhập thông qua các kiểm soát bên ngoài. Xác thực người dùng và kiểm tra tính hợp pháp của các hành động được yêu cầu, tuỳ thuộc vào quyền của người dùng, là các hoạt động căn bản. Việc bảo vệ bên trong bao gồm 3 cơ chế cơ bản sau:

1) Xác thực (*authentication*): Cơ chế này ngăn chặn người dùng trái phép, bằng cách sử dụng một hệ thống kiểm tra định danh người dùng, từ:

- Những thứ đã quen thuộc với người dùng (chẳng hạn như mật khẩu, mã);
- Những thứ mà người dùng sở hữu (chẳng hạn như thẻ từ, phù hiệu);
- Các đặc điểm vật lý của người dùng (chẳng hạn như dấu vân tay, chữ ký, giọng nói);

2) Các kiểm soát truy nhập (*access controls*): Với kết quả xác thực là hợp lệ, các câu truy vấn (do người dùng gõ vào) có thể được trả lời hay không, tuỳ thuộc vào các quyền mà người dùng hiện có.

3) Các cơ chế kiểm toán (*auditing mechanisms*): giám sát việc sử dụng tài nguyên hệ thống của người dùng. Các cơ chế này bao gồm hai giai đoạn:

- Giai đoạn đăng nhập: tất cả các câu hỏi truy nhập và câu trả lời liên quan đều được ghi lại;
- Giai đoạn báo cáo: các báo cáo của giai đoạn trước được kiểm tra, nhằm phát hiện các xâm phạm hoặc tấn công có thể xảy ra.

Các cơ chế kiểm toán thích hợp cho việc bảo vệ dữ liệu, bởi vì chúng hỗ trợ:

- Đánh giá phản ứng của hệ thống đối với một số dạng hiểm họa. Điều này cũng giúp ích cho việc phát hiện các điểm yếu của hệ thống.

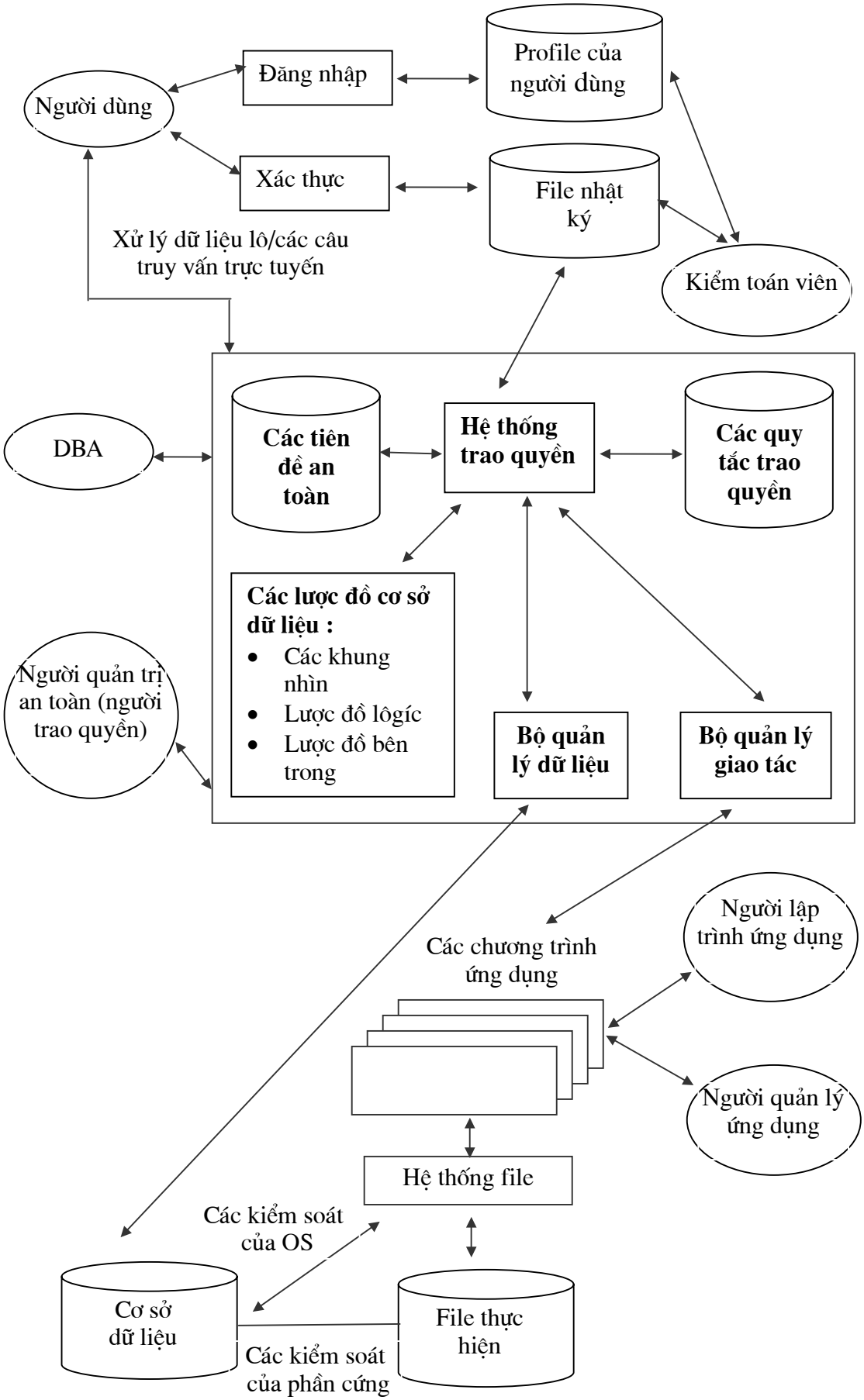
- Phát hiện các xâm phạm chủ ý được thực hiện thông qua chuỗi các câu truy vấn.

Hơn nữa, có thể ngăn chặn được các xâm phạm hoặc hiểm hoạ, do người dùng đã có ý thức trong việc sử dụng các thủ tục kiểm toán (có khả năng giám sát mọi hoạt động).

Hình 8 minh hoạ cấu trúc của một DBMS có các đặc tính an toàn, với các môđun và người dùng. Giả thiết rằng, việc truy nhập dữ liệu được bảo vệ chỉ thông qua các chức năng của DBMS. Sau khi người dùng đăng nhập và được xác thực, mỗi câu hỏi truy nhập cơ sở dữ liệu (được tạo ra từ một trình ứng dụng) được dàn xếp, thông qua các thủ tục của hệ thống trao quyền (AS). Chúng tham chiếu vào các file quy tắc trao quyền, kiểm tra xem các câu hỏi có tuân theo các quy tắc đó không. Truy nhập được phép phụ thuộc vào việc đối chiếu câu hỏi- quy tắc. Mặt khác, một thông báo về tình trạng lỗi có thể được gửi đến người dùng, và/ hoặc các xâm phạm được AS ghi trong một file nhật ký cùng với các tham chiếu (ví dụ ngày, giờ, người dùng). Người có trách nhiệm sẽ kiểm tra file này một cách định kỳ, phát hiện ra các hành vi đáng ngờ, hoặc kiểm tra tình trạng tái diễn của các xâm phạm.

Một chuyên gia, người quản trị an toàn có trách nhiệm định nghĩa các quy tắc trao quyền, xuất phát từ các yêu cầu an toàn của tổ chức. Người trao quyền cũng có thể là người kiểm toán và/ hoặc DBA.

Các câu hỏi truy nhập (được phép) được biên dịch thành các lời gọi chương trình từ thư viện ứng dụng, sau đó được bộ quản lý giao tác xử lý và chuyển thành các yêu cầu truy nhập dữ liệu (do bộ quản lý dữ liệu xử lý). Hơn nữa, hệ điều hành và phần cứng có thể đưa ra các kiểm soát (chẳng hạn như kiểm soát truy nhập file) để đảm bảo rằng dữ liệu được chuyển chính xác tới vùng người dùng yêu cầu. Kỹ thuật mật mã và các bản sao dự phòng là phương tiện thường được sử dụng khi bảo vệ hệ thống lưu giữ chương trình và dữ liệu vật lý.



Hình 8. Kiến trúc của một DBMS có các đặc tính an toàn

Mô đun an toàn của DBMS quản lý tất cả các câu hỏi. Nó gồm có các quy tắc trao quyền (cho kiểm soát tuỳ ý) và các tiên đề an toàn (cho các kiểm soát bắt buộc). AS sử dụng một, hoặc cả hai, tuỳ thuộc vào các chính sách bảo vệ của hệ thống. Trong cùng một môđun có thể có nhiều lược đồ DB, vì chúng cũng là các đối tượng cần được bảo vệ.

Quản lý hệ thống an toàn có các vai trò sau:

- Người quản lý ứng dụng: có trách nhiệm đối với việc phát triển và duy trì, hoặc các chương trình thư viện;
- DBA: quản lý các lược đồ khái niệm và lược đồ bên trong của cơ sở dữ liệu;
- Nhân viên an toàn: xác định các quyền truy nhập, và/hoặc các tiên đề, thông qua các quy tắc trong một ngôn ngữ thích hợp (có thể là DLL, hoặc DML);
- Kiểm toán viên: chịu trách nhiệm kiểm tra các yêu cầu kết nối và các câu hỏi truy nhập, nhằm phát hiện ra các xâm phạm quyền.

5. Thiết kế cơ sở dữ liệu an toàn

Như chúng ta đã biết, an toàn cơ sở dữ liệu bao gồm:

(1) Mức ngoài: kiểm soát truy nhập vật lý vào hệ thống xử lý, bảo vệ hệ thống xử lý khỏi các thảm hoạ tự nhiên, do con người hoặc máy móc gây ra.

(2) Mức trong: chống lại các tấn công có thể xảy ra đối với hệ thống, xuất phát từ sự không trung thực, gây lỗi hoặc thiếu tinh thần trách nhiệm của những người trong hoặc bên ngoài hệ thống.

Các mức này được xem là *an toàn vật lý* và *an toàn lôgíc*. Vài năm trước đây, an toàn vật lý được chú ý nhiều hơn cả, vì xét theo góc độ bảo vệ chung, nó là một "quá trình khoá" tài nguyên hệ thống trong môi trường vật lý an toàn. Không thể đảm bảo an toàn chắc chắn nếu chỉ dựa vào bảo vệ vật lý. Trong thực tế, người dùng hợp pháp có thể truy nhập gian lận dữ liệu. Đây là một hình thức lạm dụng quyền. Do vậy, quyền truy nhập thông tin "nhậy cảm" chỉ nên trao cho những người dùng được chọn lựa, tùy thuộc vào chế độ truy nhập đã chọn và tập giới hạn các mục dữ liệu.

Nói chung, các yêu cầu bảo vệ của một hệ thống gắn liền với môi trường (nơi hệ thống được sử dụng) và tình trạng kinh tế. Các đặc tính an toàn làm tăng chi phí và giảm hiệu năng. Chúng còn làm tăng độ phức tạp của hệ thống, làm giảm tính mềm dẻo, đòi hỏi nguồn nhân lực cho việc thiết kế, quản lý và duy trì, tăng yêu cầu đối với phần mềm và phần cứng. Tuy nhiên, hiện nay chúng ta còn thiếu các biện pháp an toàn thông qua phát hiện các rủi ro có chi phí khắc phục hệ thống hỏng rất lớn. Cần đánh giá chính xác các sự rủi ro, dựa trên loại hình môi trường và người dùng. Ví dụ, các yêu cầu an toàn của các hệ thống thông tin thương mại/ cá nhân và hệ thống thông tin của chính phủ có sự khác nhau.

5.1 Cơ sở dữ liệu trong các cơ quan chính phủ

Tại một số nước, sau khi phân tích các vấn đề về an toàn cơ sở dữ liệu, người ta đã tiến hành phân loại một số cơ sở dữ liệu, tuỳ thuộc vào nội dung của chúng: thông tin thiết yếu là thông tin cần thiết cho an ninh quốc gia và thông tin không thiết yếu là thông tin được biết, dựa vào các kiểm soát hoặc quyền thích hợp.

Cơ sở dữ liệu có các kiểu thông tin này được gọi là các cơ sở dữ liệu được phân loại. Trong đó, dữ liệu được phân thành các mức an toàn khác nhau (chẳng hạn như mật, tuyệt mật), tuỳ theo mức nhạy cảm của nó. Truy nhập được trao chính xác cho người dùng và giao dịch, tuỳ theo mức an toàn định trước.

Trong những môi trường này, việc phát hiện các nỗ lực thâm nhập rất khó khăn. Động cơ thâm nhập vào cơ sở dữ liệu của các cá nhân cao hơn, họ có thể sử dụng các công cụ tinh vi không để lại dấu vết. Tính toàn vẹn thông tin và từ chối dịch vụ (ngăn chặn người dùng hợp pháp sử dụng tài nguyên hệ thống) là các vấn đề trong kiểu cơ sở dữ liệu này.

5.2 Các cơ sở dữ liệu thương mại

Trước hết, việc đánh giá thiệt hại trong các hệ thống thông tin của các tổ chức thương mại khá dễ dàng. Mức độ nhạy cảm của dữ liệu do tổ chức công bố, bằng cách phân biệt giữa dữ liệu thiết yếu và dữ liệu có yêu cầu bảo vệ thấp hơn. Do vậy, thiết kế an toàn trong các cơ sở dữ liệu thương mại rất ít khi được xem là mối quan tâm hàng đầu, các vấn đề an toàn cũng không được chú ý nhiều.

Trong các môi trường này, các vấn đề an toàn xuất phát từ người dùng hợp pháp; Thực tế, việc kiểm tra sơ bộ độ tin cậy của người dùng còn lỏng lẻo. Các thủ tục

trao quyền chưa thích hợp, các kỹ thuật kiểm soát và công cụ kiểm tra truy nhập (vào dữ liệu và chương trình) mà người dùng được phép, còn khá nghèo nàn.

Hơn nữa, độ phức tạp của các vấn đề an toàn phụ thuộc vào ngữ nghĩa của cơ sở dữ liệu. Độ an toàn do công nghệ DBMS cung cấp hiện nay khá thấp. Thực tế, cơ sở dữ liệu là điểm yếu dễ bị tấn công bởi các tấn công đơn giản, chứ chưa nói đến các kỹ thuật phức tạp như con ngựa thành tơ roa, tấn công suy diễn, sâu, các trình tìm vết và cửa sập.

Các kiến trúc DBMS an toàn đa mức đã được đề xuất, nhằm đáp ứng các yêu cầu bảo vệ đa mức. Một số kiến trúc đa mức được đề xuất là Integrity Lock, Kernelized, Replicated. Chúng ta sẽ xem xét chi tiết các kiến trúc này trong các chương sau.

Tóm lại

Kiểm soát truy nhập trong một hệ thống tuân theo các chính sách truy nhập (chỉ ra ai là người có thể truy nhập và truy nhập vào những đối tượng nào của hệ thống).

Chính sách truy nhập không nên phụ thuộc vào các cơ chế thực thi kiểm soát truy nhập vật lý. Chính sách truy nhập xác định các yêu cầu truy nhập. Sau đó, các yêu cầu được chuyển thành các quy tắc truy nhập, dựa vào các chính sách được phê chuẩn. Đây là giai đoạn thiết yếu khi phát triển hệ thống an toàn. Tính đúng đắn và đầy đủ của các quy tắc và cơ chế thực thi tương ứng được xác định trong giai đoạn này. Quá trình ánh xạ cần được thực hiện, bằng cách sử dụng các kỹ thuật xây dựng mô hình cho các yêu cầu và chính sách an toàn: một mô hình cho phép nhà thiết kế miêu tả rõ ràng và kiểm tra các đặc tính an toàn của hệ thống.

Có rất nhiều hiểm hoạ có thể xảy ra đối với tính bí mật và toàn vẹn của cơ sở dữ liệu, chúng làm cho việc bảo vệ cơ sở dữ liệu trở nên phức tạp hơn. Chính vì vậy, việc bảo vệ cơ sở dữ liệu đòi hỏi nhiều biện pháp, trong đó có cả con người, phần mềm và phần cứng. Bất kỳ điểm yếu nào của chúng cũng làm ảnh hưởng đến độ an toàn của toàn bộ hệ thống. Hơn nữa, bảo vệ dữ liệu cũng nảy sinh nhiều vấn đề về tính tin cậy của hệ thống.

Tóm lại, khi phát triển một hệ thống an toàn, chúng ta cần quan tâm đến một số khía cạnh thiết yếu sau:

- Các đặc điểm của môi trường xử lý và lưu giữ thực tế. Cần phân tích cẩn thận để định ra mức bảo vệ theo yêu cầu của hệ thống: đây chính là các yêu cầu an toàn;
- Các cơ chế bảo vệ bên ngoài môi trường xử lý. Chúng là các biện pháp kiểm soát vật lý và quản trị, góp phần đảm bảo hiệu lực của các cơ chế hoạt động an toàn;
- Các cơ chế bảo vệ cơ sở dữ liệu bên trong. Chúng được thực hiện sau khi người dùng qua được các kiểm soát đăng nhập và xác thực;
- Tổ chức vật lý của các thông tin được lưu giữ;
- Các đặc tính an toàn do hệ điều hành và phần cứng cung cấp;
- Độ tin cậy của phần mềm và phần cứng;
- Các khía cạnh về tổ chức, con người.

THIẾT KẾ CƠ SỞ DỮ LIỆU AN TOÀN

1 Giới thiệu

An toàn cơ sở dữ liệu lôgíc giải quyết các vấn đề an toàn (tính bí mật và toàn vẹn) thông qua một bộ các quy tắc nhằm thiết lập các truy nhập hợp pháp vào thông tin và tài nguyên của cơ sở dữ liệu. Các quy tắc này phải được định nghĩa chính xác và dựa trên cơ sở (hay tuân theo) các yêu cầu và chính sách an toàn của tổ chức, tránh các mâu thuẫn và các lỗi có thể là các điểm yếu dễ bị tấn công của hệ thống. An toàn lôgíc phải được coi là một phần bên trong của hệ thống an toàn toàn cục của tổ chức.

Thiết kế lôgíc của một hệ thống an toàn có nghĩa là thiết kế phần mềm an toàn và các quy tắc an toàn. Phần mềm an toàn bao gồm các bó chương trình an toàn, chẳng hạn như các hệ điều hành an toàn, các DBMS an toàn và các thủ tục an toàn phi thể thức. Thiết kế phải tận dụng được các chuẩn an toàn hiện có. Các quy tắc an toàn phải được định nghĩa chính xác và thích ứng, đáp ứng được các yêu cầu khác nhau của người sử dụng và đảm bảo tính bí mật và toàn vẹn của một hệ thống. Thêm vào đó, việc thiết kế các quy tắc an toàn phải phù hợp với các hoạt động thiết kế cơ sở dữ liệu.

Các quy tắc an toàn đối với một cơ sở dữ liệu ngày càng phức tạp hơn, nó không chỉ đơn thuần là các danh sách kiểm soát truy nhập, hoặc các bảng biểu đơn giản. Trong thực tế, cơ sở của các quy tắc an toàn có thể được xem như là một cơ sở dữ liệu.

Nói chung, các giai đoạn phân tích, thiết kế khái niệm, thực hiện thiết kế chi tiết, thử nghiệm và duy trì cũng được áp dụng khi phát triển hệ thống an toàn.

Mục đầu tiên của phần này trình bày các giải pháp được sử dụng để thiết kế DBMS an toàn; Trình bày một số mẫu nghiên cứu và các sản phẩm DBMS an toàn thương mại; Tiếp theo trình bày một giải pháp mang tính phương pháp luận nhằm thiết kế các quy tắc an toàn.

2 Thiết kế DBMS an toàn

Cơ sở dữ liệu là một tập hợp các dữ liệu được tổ chức và quản lý thông qua phần mềm xác định, DBMS.

Việc đảm bảo an toàn cơ sở dữ liệu thông qua các kỹ thuật ở cả hai mức DBMS và OS. Khi thực hiện các yêu cầu an toàn, DBMS có thể khai thác các chức năng an toàn bắt buộc ở mức OS. Nói riêng, các chức năng quản lý I/O và quản lý tài nguyên chia sẻ chứng minh tính an toàn của các môi trường DBMS. Tuy nhiên, các chức năng an toàn DBMS không nên bị coi là một mở rộng của các chức năng OS cơ sở. Các mối quan tâm khác nhau về an toàn giữa các OS và DBMS được liệt kê sau đây:

- Độ chi tiết của đối tượng (*Object granularity*): Trong OS, độ chi tiết ở mức tệp (file). Trong DBMS , nó chi tiết hơn (ví dụ như: các quan hệ, các hàng, các cột, các trường).

- Các tương quan ngữ nghĩa trong dữ liệu (*Semantic correlations among data*): Dữ liệu trong một cơ sở dữ liệu có ngữ nghĩa và liên quan với nhau thông qua các quan hệ ngữ nghĩa. Do vậy, nên tuân theo các kiểu kiểm soát truy nhập khác nhau, tuỳ thuộc vào các nội dung của đối tượng, ngữ cảnh và lược sử truy nhập, để bảo đảm thực hiện chính xác các yêu cầu an toàn trong dữ liệu.

- Siêu dữ liệu (*Metadata*): Siêu dữ liệu tồn tại trong một DBMS, cung cấp thông tin về cấu trúc của dữ liệu trong cơ sở dữ liệu. Siêu dữ liệu thường được lưu giữ trong các từ điển dữ liệu. Ví dụ, trong các cơ sở dữ liệu, siêu dữ liệu mô tả các thuộc tính, miền của các thuộc tính, quan hệ giữa các thuộc tính và vị trí phân hoạch cơ sở dữ liệu. Trong thực tế, siêu dữ liệu có thể cung cấp các thông tin nhạy cảm về nội dung của cơ sở dữ liệu (các kiểu dữ liệu và quan hệ) và có thể được sử dụng như là một phương pháp nhằm kiểm soát truy nhập vào dữ liệu cơ sở. Không có các mô tả siêu dữ liệu tồn tại trong OS.

- Các đối tượng lôgíc và vật lý (*Logical and physical objects*): Các đối tượng trong một OS là các đối tượng vật lý (ví dụ: các file, các thiết bị, bộ nhớ và các tiến trình). Các đối tượng trong một DBMS là các đối tượng lôgíc (ví dụ: các quan hệ, các khung nhìn). Các đối tượng lôgíc của DBMS không phụ thuộc vào các đối tượng vật lý của OS, điều này đòi hỏi các yêu cầu và các kỹ

thuật an toàn đặc biệt, được định hướng cho việc bảo vệ đối tượng của cơ sở dữ liệu.

- Các kiểu dữ liệu bội (*Multiple data types*): Đặc điểm của các cơ sở dữ liệu được xác định thông qua các kiểu dữ liệu, cho các chế độ đa truy nhập nào được yêu cầu (ví dụ: chế độ thống kê, chế độ quản trị). Tại mức OS chỉ tồn tại truy nhập vật lý, cho ghi, đọc và thực hiện các thao tác.

- Các đối tượng động và tĩnh (*Static and dynamic objects*): Các đối tượng được OS quản lý là các đối tượng tĩnh và tương ứng với các đối tượng thực. Trong các cơ sở dữ liệu, các đối tượng có thể được tạo ra động (ví dụ, các kết quả hỏi đáp) và không có các đối tượng thực tương ứng. Ví dụ, khung nhìn của các cơ sở dữ liệu được tạo ra động, như các quan hệ ảo có nguồn gốc từ các quan hệ cơ sở được lưu giữ thực tế trong cơ sở dữ liệu. Nên định nghĩa các yêu cầu bảo vệ xác định nhằm đối phó với các đối tượng động.

- Các giao tác đa mức (*Multilevel transactions*): Trong một DBMS thường có các giao tác đòi hỏi dữ liệu ở các mức an toàn khác nhau. DBMS phải bảo đảm các giao tác đa mức được thực hiện theo một cách an toàn. Tại mức OS, chỉ có các thao tác cơ bản mới được thực hiện (ví dụ, đọc, ghi, thực hiện), đòi hỏi dữ liệu có cùng mức an toàn.

- Thời gian tồn tại của dữ liệu (*Data life cycle*): Dữ liệu trong một cơ sở dữ liệu có thời gian tồn tại dài và DBMS có thể đảm bảo việc bảo vệ từ đầu đến cuối thời gian tồn tại của dữ liệu.

2.1. Các cơ chế an toàn trong các DBMS

An toàn dữ liệu được quan tâm cùng với việc khám phá hoặc sửa đổi trái phép thông tin, chẳng hạn như chèn thêm các mục dữ liệu, xoá, thay đổi dữ liệu hiện có. Một DBMS đòi hỏi nhiều tính năng nhằm đạt được các yêu cầu an toàn của một hệ thống thông tin. DBMS đóng một vai trò trung tâm bởi vì nó xử lý các quan hệ phức tạp trong dữ liệu. Một số chức năng an toàn chủ chốt phải được OS cung cấp, trong khi đó các ràng buộc an toàn xác định tại mức ứng dụng lại được DBMS xử lý, sau đó nó được uỷ thác ngăn chặn các ứng dụng khám phá hoặc làm hư hại dữ liệu.

Các yêu cầu an toàn chính (mà một DBMS nên cung cấp) liên quan đến các vấn đề sau đây:

- Các mức độ truy nhập chi tiết khác nhau (*Different degrees of granularity of access*): DBMS nên bảo đảm kiểm soát truy nhập với các mức độ chi tiết khác nhau. Đối với các cơ sở dữ liệu, kiểm soát truy nhập có thể được áp dụng theo các mức độ chi tiết như: cơ sở dữ liệu, các quan hệ, một quan hệ, một số cột của một quan hệ, một số hàng của một quan hệ, một số hàng và một số cột của một quan hệ.

- Các chế độ truy nhập khác nhau (*Different access modes*): Phải phân biệt được các kiểm soát truy nhập (ví dụ, một người làm công được quyền đọc một mục dữ liệu nhưng không được phép ghi lên nó). Trong các cơ sở dữ liệu, các chế độ truy nhập được đưa ra dưới dạng các lệnh SQL cơ bản (ví dụ: SELECT, INSERT, UPDATE, DELETE).

- Các kiểu kiểm soát truy nhập khác nhau (*Different types of access controls*): Các yêu cầu truy nhập có thể được xử lý, bằng cách sử dụng các kiểu kiểm soát khác nhau.

  - Các kiểm soát phụ thuộc tên (*Name-dependent controls*) dựa vào tên của đối tượng bị truy nhập.

  - Các kiểm soát phụ thuộc dữ liệu (*Data-dependent controls*) thực hiện truy nhập phụ thuộc vào các nội dung của đối tượng bị truy nhập.

  - Các kiểm soát phụ thuộc ngữ cảnh (*Context-dependent controls*) chấp thuận hoặc từ chối truy nhập phụ thuộc vào giá trị của một số biến hệ thống (ví dụ như: ngày, tháng, thiết bị đầu cuối yêu cầu).

  - Các kiểm soát phụ thuộc lược sử (*History-dependent controls*) quan tâm đến các thông tin về trình tự câu truy vấn (ví dụ như: các kiểu câu truy vấn, dữ liệu trả lại, profile của người dùng).

  - Các kiểm soát phụ thuộc kết quả (*Result-dependent controls*) thực hiện quyết định truy nhập phụ thuộc vào kết quả của các thủ tục kiểm soát hỗ trợ, chúng là các thủ tục được thực hiện tại thời điểm hỏi.

Trong các cơ sở dữ liệu, kiểm soát truy nhập phụ thuộc dữ liệu thường được thực hiện thông qua các cơ chế sửa đổi câu truy vấn hoặc cơ chế sửa đổi dựa vào khung nhìn. Các khung nhìn là các quan hệ ảo xuất phát từ các quan hệ cơ sở (là các quan hệ được lưu giữ thực trong cơ sở dữ liệu) và các khung nhìn khác, tuỳ thuộc vào tiêu chuẩn chọn lựa các bộ (*tuple*) hoặc các thuộc tính. Khi sử dụng một kỹ thuật sửa đổi câu truy vấn , câu truy vấn ban đầu (do người sử dụng yêu cầu) bị hạn chế đến mức nào còn tuỳ thuộc vào các quyền của người sử dụng.

- Quyền động (*Dynamic authorization*): DBMS nên hỗ trợ việc sửa đổi các quyền của người sử dụng trong khi cơ sở dữ liệu đang hoạt động. Điều này tương ứng với nguyên tắc đặc quyền tối thiểu, có thể sửa đổi các quyền tuỳ thuộc vào các nhiệm vụ của người sử dụng.

- Bảo vệ đa mức (*Multilevel protection*): Khi được yêu cầu, DBMS nên tuân theo bảo vệ đa mức, thông qua chính sách bắt buộc. Các kiểm soát truy nhập bắt buộc (*Mandatory access controls*) dựa vào các nhãn an toàn được gán cho các đối tượng (là các mục dữ liệu) và các chủ thể (là những người sử dụng). Trong các môi trường quân sự, các nhãn an toàn bao gồm: một thành phần phân cấp (*hierarchical component*) và một tập rỗng các loại không phân cấp (có thể có). DBMS nên cung cấp các kỹ thuật để định nghĩa các nhãn an toàn và gán nhãn cho các đối tượng và các chủ thể. Bằng cách sử dụng các nhãn an toàn, DBMS nên tuân theo bảo vệ đa mức, trong đó các nhãn an toàn khác nhau được gán cho các mục dữ liệu khác nhau trong cùng một đối tượng. Ví dụ, trong các cơ sở dữ liệu quan hệ, mỗi quan hệ nên có một nhãn an toàn cho trước, nhãn an toàn này chứa các thuộc tính với các nhãn an toàn của chúng (có thể khác với nhãn quan hệ) và lần lượt lưu giữ các giá trị với các nhãn an toàn của chúng.

- Các kênh ngầm (*Covert channels*): DBMS không nên để người sử dụng thu được các thông tin trái phép thông qua các phương pháp truyền thông gián tiếp.

- Các kiểm soát suy diễn (*Inference controls*): Các kiểm soát suy diễn nên ngăn chặn người sử dụng suy diễn dựa vào các thông tin mà họ thu được trong cơ sở dữ liệu. Trong một hệ thống cơ sở dữ liệu, các vấn đề suy diễn thường liên quan đến các vấn đề về gộp (*aggregation*) và quan hệ dữ liệu

(*data association*). DBMS nên cung cấp một cách thức để gán phân loại cho các thông tin được gộp, phản ánh mức nhạy cảm của các mục dữ liệu được gộp. Khi đó, các thông tin (như quan hệ của các mục dữ liệu, hoặc tập hợp các mục dữ liệu) nhạy cảm hơn so với các mục dữ liệu đơn lẻ, nên chúng phải được quản lý phù hợp. DBMS không nên để người sử dụng (thông qua các kiểm soát suy diễn) biết các thông tin tích luỹ được phân loại ở mức cao, bằng cách sử dụng các mục dữ liệu được phân loại ở mức thấp. Trong một cơ sở dữ liệu quan hệ, các kỹ thuật hạn chế câu truy vấn thường được sử dụng để tránh suy diễn. Các kỹ thuật như vậy có thể sửa đổi câu truy vấn ban đầu, hoặc huỷ bỏ câu truy vấn. Các kỹ thuật đa thể hiện (*polyinstantiation*) và kiểm toán cũng có thể được sử dụng cho mục đích này. Cuối cùng, một kiểu suy diễn đặc biệt xảy ra trong các cơ sở dữ liệu thống kê, nơi mà người sử dụng không được phép suy diễn các mục dữ liệu cá nhân từ dữ liệu kiểm toán đã được gộp, người sử dụng có thể thu được các dữ liệu này từ các câu truy vấn kiểm toán.

- Đa thể hiện (*polyinstantiation*): Kỹ thuật này có thể được DBMS sử dụng để ngăn chặn suy diễn, bằng cách cho phép cơ sở dữ liệu có nhiều thể hiện cho cùng một mục dữ liệu, mỗi thể hiện có một mức phân loại riêng. Trong một cơ sở dữ liệu quan hệ có thể có các bộ khác nhau với cùng một khoá, với mức phân loại khác nhau, ví dụ nếu tồn tại một hàng (được phân loại ở mức cao) và một người sử dụng (được phân loại ở mức thấp) yêu cầu chèn thêm một hàng mới có cùng khoá. Điều này ngăn chặn người sử dụng (được phân loại ở mức thấp) suy diễn sự tồn tại của hàng (được phân loại ở mức cao) trong cơ sở dữ liệu.

- Kiểm toán (*Auditing*): Các sự kiện liên quan tới an toàn (xảy ra trong khi hệ thống cơ sở dữ liệu đang hoạt động) nên được ghi lại và theo một khuôn dạng có cấu trúc, chẳng hạn như: nhật ký hệ thống, vết kiểm toán. Các vết kiểm toán rất hữu ích cho các phân tích về sau để phát hiện ra các mối đe doạ có thể xảy ra cho cơ sở dữ liệu. Thông tin kiểm toán cũng hữu ích cho kiểm soát suy diễn, nhờ đó chúng ta có thể kiểm tra được lược sử của các câu truy vấn do người sử dụng đưa ra, để quyết định xem có nên trả lời câu truy vấn mới hay không, vì câu truy vấn mới này lại liên quan đến các đáp ứng của các câu truy vấn trước đó, có thể dẫn đến một vi phạm suy diễn.

- Các kiểm soát luồng (*Flow controls*): Các kiểm soát luồng kiểm tra đích của đầu ra. Chúng ta có thể có được kiểm soát này thông qua một truy nhập được phép (*authorized access*).

- Không cửa sau (*No back door*): Truy nhập vào dữ liệu chỉ nên xảy ra thông qua DBMS. Phải đảm bảo không có các đường dẫn truy nhập ẩn.

- Tính chất không thay đổi của các cơ chế (*Uniformity of mechanisms*): Nên sử dụng các cơ chế chung để hỗ trợ các chính sách khác nhau và tất cả các kiểm soát liên quan tới an toàn (các kiểm soát bí mật và toàn vẹn).

- Hiệu năng hợp lý (*Reasonable performance*): Các kiểm soát an toàn làm tăng thời gian hoạt động; Cần tối thiểu hoá để đảm bảo hiệu năng của hệ thống.

Hơn nữa, ở đây có rất nhiều nguyên tắc cơ bản cho toàn vẹn thông tin, chúng độc lập với ngữ cảnh của DBMS và các đặc thù của ứng dụng. Lợi ích của các nguyên tắc này là giúp chúng ta đánh giá các chính sách an toàn của một hệ thống thông tin xác định.

- Các giao tác đúng đắn (*Well-formed transactions*): Nguyên tắc này (còn được gọi là sự thay đổi có ràng buộc) xác định: chỉ được sửa dữ liệu thông qua các giao tác đúng đắn. Độ chính xác của các giao tác này được chứng thực với một mức độ đảm bảo nào đó. Các giao tác này chuyển sang các cơ chế DBMS để đảm bảo các thuộc tính cho giao tác. DBMS phải đảm bảo rằng các cập nhật phải được thực hiện thông qua các giao tác, lưu ý rằng cơ sở dữ liệu phải được đóng gói trong DBMS thông qua OS.

- Người sử dụng được xác thực (*Authenticated users*): Theo nguyên tắc này, không nên cho người sử dụng thực hiện các thay đổi trừ khi nhận dạng của họ được xác thực chính xác. Việc xác thực người sử dụng thuộc trách nhiệm của OS và không cần phải lặp lại trong DBMS. Việc xác thực làm cơ sở cho một số nguyên tắc khác được liệt kê sau đây (đặc quyền tối thiểu, tách bạch nhiệm vụ, uỷ quyền).

- Đặc quyền tối thiểu (*Least privilege*): Đây là một nguyên tắc giới hạn người sử dụng chỉ được làm việc với một tập tối thiểu các đặc quyền và tài nguyên

cần thiết để thực hiện nhiệm vụ của mình. Đặc quyền tối thiểu chuyển sang các cơ chế DBMS cho các thao tác "đọc" và "ghi".

- Tách bạch nhiệm vụ (*Separation of duties*): Nguyên tắc này được đưa ra nhằm hạn chế tối đa một cá nhân bất kỳ có thể phá hoại dữ liệu, để đảm bảo toàn vẹn dữ liệu. Tách bạch nhiệm vụ được gắn liền với các kiểm soát trên các chuỗi giao tác. Hiện tại có nhiều cơ chế nhưng chúng không được thiết kế cho các mục đích toàn vẹn, do vậy gây ra một số bất tiện.

- Tính liên tục của hoạt động (*Continuity of operation*): Vấn đề này đã nhận được nhiều sự quan tâm, cả về mặt lý thuyết và thực tế, các giải pháp dựa trên cơ sở lặp dữ liệu cũng đã được đề xuất. Đối mặt với các sự cố phá hoại vượt ngoài tầm kiểm soát của tổ chức, nên duy trì các hoạt động của hệ thống sau khi sự cố xảy ra (quan tâm đến các biện pháp an toàn vật lý).

- Dựng lại các sự kiện (*Reconstruction of events*): Việc dựng lại các sự kiện trong một DBMS phụ thuộc vào các vết kiểm toán. Việc dựng lại có thể xảy ra ở nhiều mức vết khác nhau, nhiều việc khác nhau như: ghi lại một lược sử đầy đủ về việc sửa đổi giá trị của một mục dữ liệu, hoặc lưu giữ nhận dạng của từng cá nhân khi thực hiện từng thay đổi. Một vấn đề đặt ra là chất lượng của dữ liệu trong vết kiểm toán cũng phải phù hợp. Một số đề xuất gần đây có sử dụng các kỹ thuật của hệ chuyên gia để lưu giữ và làm sáng tỏ các vết kiểm toán.

- Kiểm tra thực tế (*Reality checks*): Việc kiểm tra định kỳ đối với các thực thể thực tế góp phần duy trì các giá trị dữ liệu chính xác trong hệ thống. DBMS có trách nhiệm duy trì một khung nhìn thích hợp bên trong cơ sở dữ liệu, làm cơ sở cho các kiểm tra bên ngoài.

- Dễ dàng sử dụng an toàn (*Ease of safe use*): Cách dễ nhất để điều hành một hệ thống cũng phải là cách an toàn nhất. Điều này có nghĩa là các thủ tục an toàn nên đơn giản, phổ biến, dễ sử dụng.

- Uỷ quyền (*Delegation of authority*): Nó quan tâm đến việc gán các đặc quyền cho tổ chức, lấy các chính sách làm cơ sở, yêu cầu các thủ tục gán phải phản ánh các quy tắc của tổ chức và chúng phải mềm dẻo. Việc uỷ quyền phải khá mềm dẻo để phù hợp với các chính sách; Nói rộng hơn, các giải pháp tổ chức

đặc thù chuyển sang các cơ chế bắt buộc và các cơ chế tuỳ ý. Trong các cơ chế tuỳ ý, việc uỷ quyền tuỳ thuộc vào bản thân chủ thể, có thể trao hoặc thu hồi, huỷ bỏ các quyền cho người sử dụng khác. Việc uỷ quyền tuân theo các chính sách tuỳ ý, hỗ trợ cấp/huỷ bỏ các quyền. Các đặc quyền đặc biệt nên tồn tại trong DBMS, trao các quyền đặc biệt này cho một số lượng hạn chế người sử dụng (có nghĩa là các đặc quyền quản trị cơ sở dữ liệu). Việc trao các quyền cho những người sử dụng khác có thể gây ra một số vấn đề, khi các quyền này bị huỷ bỏ, thu hồi. DBMS nên cung cấp các cơ chế cho việc quản lý thu hồi. Uỷ quyền là một khả năng cần thiết, nhằm phản ánh cấu trúc phân cấp của tổ chức và nên thực hiện tuân theo các quy tắc (đã được định nghĩa trong tổ chức). Uỷ quyền trong một DBMS thường tuân theo các lệnh SQL GRANT/REVOKE.

Nói riêng, khi quan tâm đến tính toàn vẹn của một DBMS, các nguyên tắc được phân nhóm như sau:

- Nhóm 1: Các giao tác đúng đắn, tính liên tục của hoạt động. Các nguyên tắc này bao trùm hoàn toàn lên các cơ chế của DBMS.

- Nhóm 2: Đặc quyền tối thiểu, tách bạch nhiệm vụ, xây dựng lại các biến cố và uỷ quyền. Nhiều cơ chế mới được yêu cầu cho nhóm này và một số giải pháp đầy triển vọng nhằm mở rộng các cơ chế của DBMS cũng được đưa ra.

- Nhóm 3: Người sử dụng được xác thực, kiểm tra thực tế và dễ dàng sử dụng an toàn. Xác thực là trách nhiệm của OS, kiểm tra thực tế tuỳ thuộc vào an toàn tổ chức.

2. 2 Mô mình cấp quyền System R

♣ Mô hình này quan tâm đến các bảng của cơ sở dữ liệu. Chúng có thể là các bảng cơ sở hoặc các bảng của khung nhìn. Chủ thể của mô hình này là người sử dụng, đây là những người có thể truy nhập vào cơ sở dữ liệu. Các đặc quyền mà mô hình này quan tâm là các chế độ truy nhập có thể được áp dụng cho các bảng của cơ sở dữ liệu. Nói riêng, nó quan tâm đến các chế độ truy nhập sau:

Read	Để đọc các bộ (các hàng) từ một bảng
Insert	Thêm các bộ vào một bảng

Delete	Xoá các bộ từ một bảng
Update	Sửa đổi nội dung các bộ hiện có trong một bảng
Drop	Xoá toàn bộ một bảng ra khỏi hệ thống

Mô hình đã hỗ trợ quản trị quyền phi tập trung (*Decentralized administration of authorizations*). Nói riêng, người sử dụng của một cơ sở dữ liệu bất kỳ có thể được phép tạo ra một bảng mới. Khi người sử dụng tạo ra một bảng mới, chỉ có anh ta mới được phép thực hiện các đặc quyền trên bảng (điều này không hoàn toàn đúng nếu bảng là một khung nhìn, chúng ta sẽ xem xét sau). Như người chủ sở hữu, người sử dụng có thể trao các đặc quyền trên bảng cho những người sử dụng khác. Khi đó, một quyền mới sẽ được chèn thêm vào tập hợp các quyền được quản lý trong hệ thống. Việc trao các đặc quyền cho người sử dụng khác có thể tuỳ chọn. Mỗi quyền có thể được mô tả như là một bộ {*s, p, t, ts,g, go*}, trong đó:

s là chủ thể (người sử dụng) được trao quyền;

p là đặc quyền (chế độ truy nhập);

t là bảng, quyền được áp dụng trên đó;

ts là thời điểm quyền được trao;

g là người đã trao quyền;

go ∈ {yes, no} chỉ báo khi nào *s* có tuỳ chọn trao *p* trên *t*

Nếu đặc quyền là "*update*" thì các cột (mà trên đó đặc quyền được thực hiện) phải được chỉ báo. Tuỳ chọn trao (*Grant option*) tương tự như là cờ *copy* của mô hình ma trận truy nhập. Nếu một người sử dụng nắm giữ một đặc quyền trên bảng với tuỳ chọn trao, người sử dụng này cũng có thể trao đặc quyền trên bảng cùng với tuỳ chọn trao cho những người sử dụng khác.

Trên mỗi quyền, người trao quyền (*g*) và thời điểm trao quyền (*ts*) cần được chỉ báo (được sử dụng cho thủ tục thu hồi).

Việc định nghĩa, thao tác dữ liệu và kiểm soát ngôn ngữ của System R, được đặt tên là SQL, trong đó có các lệnh cho phép người sử dụng yêu cầu thực hiện các thao tác trao và thu hồi. Lệnh trao của SQL có dạng như sau:

GRANT {ALL RIGHT (privileges) ALL BUT (privileges)} ON (table) TO (user-list) [WITH GRANT OPTION]

Người sử dụng (người trao đặc quyền trên một bảng) cũng có thể ghi rõ từ khoá PUBLIC, thay cho (user-list). Khi đó, tất cả những người sử dụng của cơ sở dữ liệu đều được trao đặc quyền trên bảng.

Những người sử dụng (người có đặc quyền trên một bảng với tuỳ chọn trao) cũng có thể thu hồi đặc quyền trên bảng. Tuy nhiên, anh ta chỉ có thể thu hồi các quyền mà anh ta đã trao. Lệnh thu hồi của SQL có dạng như sau:

REVOKE {ALL RIGHTS (privileges)} ON (table) FROM (user-list)

♣ *Đối với việc thu hồi quyền, mô hình quyền System R sử dụng cơ chế thu hồi đệ quy. Chúng ta có thể diễn giải như sau: người sử dụng* x *thu hồi đặc quyền* p *trên bảng* t *từ người sử dụng* y.

♣ Các khung nhìn

Mô hình System R cho phép người sử dụng định nghĩa các khung nhìn ở trên các bảng cơ sở và các khung nhìn khác. Các khung nhìn (được định nghĩa trong các giới hạn của các câu truy vấn có trên một hoặc nhiều bảng cơ sở hoặc các khung nhìn) tương ứng với một cơ chế đơn lẻ và có hiệu lực để hỗ trợ cho các quyền phụ thuộc nội dung.

Người sử dụng (người định nghĩa một khung nhìn) là chủ sở hữu của khung nhìn. Tuy nhiên, chưa chắc anh ta đã được phép thực hiện tất cả các đặc quyền trên khung nhìn. Các quyền mà người sở hữu khung nhìn có được trên khung nhìn phụ thuộc vào ngữ nghĩa của khung nhìn (có thể có một số thao tác nào đó không có khả năng được thực hiện trên khung nhìn) và phụ thuộc vào các quyền mà người sử dụng có được trên các bảng có khung nhìn tham chiếu trực tiếp vào các bảng này. Nếu khung nhìn được định nghĩa trên một bảng đơn lẻ, người sử dụng có thể được phép thực hiện tất cả các đặc quyền trên bảng. Nếu khung nhìn được định nghĩa trên một tập hợp các bảng, người sử dụng được phép thực hiện tất cả các đặc quyền

mà anh ta có trên mọi bảng được khung nhìn tham chiếu trực tiếp. Tuy nhiên, các đặc quyền trên khung nhìn có thể bị hạn chế hơn, phụ thuộc vào ngữ nghĩa của khung nhìn.

Các đặc quyền trên khung nhìn của người sở hữu được xác định tại thời điểm định nghĩa khung nhìn. Với mọi đặc quyền mà người sử dụng có được trên tất cả các bảng mà khung nhìn tham chiếu trực tiếp, các quyền tương ứng trên khung nhìn được định nghĩa. Nếu người sử dụng (người định nghĩa khung nhìn) được phép thực hiện một đặc quyền trên tất cả các bảng cơ sở với tuỳ chọn trao, người sử dụng sẽ được cho trước tuỳ chọn trao dành cho đặc quyền trên khung nhìn. Nhãn thời gian chỉ báo thời điểm định nghĩa khung nhìn.

Nếu người sử dụng nhận được một đặc quyền trên một khung nhìn với tuỳ chọn trao, anh ta có thể trao đặc quyền cho những người sử dụng khác, có thể có tuỳ chọn trao.

Sau khi có một khung nhìn đã được định nghĩa, nếu người sở hữu khung nhìn nhận thêm các đặc quyền trên các bảng cơ sở, các đặc quyền này sẽ không được áp dụng trên khung nhìn; có nghĩa là người sử dụng sẽ không được phép sử dụng chúng trên khung nhìn. Ngược lại, nếu sau khi định nghĩa một khung nhìn, người sở hữu khung nhìn bị huỷ bỏ một đặc quyền trên bất kỳ bảng cơ sở nào, cũng cần huỷ bỏ đặc quyền trên khung nhìn.

♣ Việc thực hiện mô hình

Các thông tin về quyền truy nhập cơ sở dữ liệu của người sử dụng được lưu giữ trong 2 quan hệ, chúng có tên là SYSAUTH và SYCOLAUTH. Quan hệ SYSAUTH có các thuộc tính sau:

Userid	Chỉ ra người sử dụng
Tname	Chỉ ra bảng có quyền tham chiếu vào
Type	Chỉ ra kiểu của bảng Tname. Thuộc tính có giá trị "R" nếu bảng là một bảng cơ sở và "V" nếu bảng là một bảng của khung nhìn
Grantor	Chỉ ra người trao các quyền

Read Chỉ ra thời điểm Grantor trao đặc quyền đọc trên bảng cho Grantee. Nếu grantor không trao đặc quyền này cho grantee, thuộc tính có giá trị 0

Insert Chỉ ra thời điểm grantor trao đặc quyền chèn thêm trên bảng cho grantee. Nếu grantor không trao đặc quyền này cho grantee, thuộc tính có giá trị 0

Delete Chỉ ra thời điểm grantor trao đặc quyền cập nhật trên bảng cho grantee. Nếu grantor không trao đặc quyền này cho grantee, thuộc tính có giá trị 0

Update Chỉ ra các cột có đặc quyền cập nhật được trao. Nếu có giá trị "All", có nghĩa là có thể cập nhật trên tất cả các cột. Nếu có giá trị "None", có nghĩa là không có đặc quyền cập nhật trên bảng, hoặc "Some", có nghĩa là đặc quyền chỉ được thực hiện trên một số cột nào đó của bảng

Grantopt Chỉ ra khi có các đặc quyền được trao với tùy chọn trao

Quan hệ SYCOLAUTH có các thuộc tính sau đây:

Userid Chỉ ra người sử dụng được trao đặc quyền cập nhật

Table Chỉ ra bảng trên đó đặc quyền cập nhật được trao

Column Chỉ ra cột của Table trên đó đặc quyền cập nhật được trao

Grantor Chỉ ra người sử dụng đã trao đặc quyền

Grantopt Chỉ ra khi có đặc quyền được trao với tuỳ chọn trao

♣ Các mở rộng cho mô hình

Mô hình quyền của System R đã được Wilms và Linsday mở rộng năm 1982 với nhiều chức năng phục vụ cho quản lý nhóm. Người sử dụng có thể được phân thành các nhóm. Các nhóm có thể gắn kết với nhau, một nhóm có thể xuất hiện như là một thành viên của nhóm khác. Sau đó, các quyền truy nhập có thể được trao cho một nhóm, có nghĩa là các quyền này được trao cho tất cả các thành viên của nhóm.

Hai mở rộng chính cho mô hình được đưa ra như sau.

Mở rộng thứ nhất giới thiệu một kiểu thao tác thu hồi, trong đề xuất ban đầu, bất cứ khi nào xảy ra việc thu hồi một quyền từ một người sử dụng, có thể phải thực hiện quá trình thu hồi đệ quy. Một vấn đề xảy ra đối với giải pháp này là nó rất dễ bị phá vỡ. Thật vậy, trong nhiều tổ chức, các quyền (mà một người sử dụng sở hữu) liên quan đến nhiệm vụ hoặc chức năng đặc thù của anh ta trong tổ chức. Nếu người sử dụng thay đổi nhiệm vụ hoặc chức năng của anh ta, ví dụ, anh ta được thăng chức; Làm sao có thể loại bỏ các quyền dành người sử dụng này mà không cần phải thu hồi đệ quy tất cả các quyền mà người sử dụng này đã trao. Vì vậy, có một kiểu thao tác thu hồi không cần thực hiện quá trình thu hồi đệ quy các quyền.

Mở rộng thứ hai quan tâm đến quyền phủ định. Hầu hết các DBMS sử dụng chính sách thế giới khép kín. Theo chính sách này, việc thiếu vắng một quyền được hiểu như là một quyền phủ định. Do vậy, bất kỳ khi nào người sử dụng cố gắng truy nhập vào một đối tượng, nếu không tìm thấy quyền hợp lệ trong các catalog của hệ thống, người sử dụng không được phép truy nhập. Giải pháp này có một vấn đề chính, đó là sự thiếu vắng một quyền xác định (dành cho một người sử dụng xác định) không ngăn chặn được việc anh ta nhận được quyền này ngay sau đó. Quyền phủ định thường mạnh hơn quyền khẳng định (các quyền được phép truy nhập). Vì vậy, bất kỳ khi nào người sử dụng có cả 2 quyền (khẳng định và phủ định) trên cùng một đối tượng, người sử dụng không được phép truy nhập vào đối tượng, ngay cả khi quyền khẳng định được trao ngay sau khi quyền phủ định được trao, người sử dụng vẫn không được phép truy nhập.

2.3 Các kiến trúc của DBMS an toàn

Trong phần này trình bày một số đặc điểm chính của của các kiến trúc DBMS an toàn. Các DBMS an toàn hoạt động theo 2 chế độ: mức an toàn hệ thống cao và đa mức.

Trong các DBMS mức an toàn hệ thống cao, tất cả những người sử dụng được chuyển sang mức an toàn cao nhất, trước khi loại bỏ dữ liệu và có một người có trách nhiệm xem xét dữ liệu này để loại bỏ chúng một cách chính xác. Giải pháp này cho phép người sử dụng sử dụng các kỹ thuật DBMS hiện có, nhưng phát sinh một số chi phí cho các "thủ tục cho phép" và xem xét dữ liệu thủ công. Chế độ này có thể làm tăng thêm một số rủi ro an toàn khi tất cả những người sử dụng được chuyển sang mức cho phép cao nhất.

Với chế độ đa mức, có thể có nhiều kiểu kiến trúc khác nhau, dựa vào việc sử dụng các DBMS tin cậy và không tin cậy. Các kiến trúc đa mức như: kiến trúc *Trusted Subject (chủ thể tin cậy)* và các kiến trúc *Woods Hole*, chúng được Woods Hole Summer Study đề xuất năm 1982. Các kiến trúc Woods Hole bao gồm: *Integrity Lock*, *Kernelized* và các kiến trúc *Replicated*. Trong các kiến trúc Trusted Subject, sử dụng cả DBMS tin cậy và DBMS không tin cậy, trong khi đó các kiến trúc Woods Hole chỉ sử dụng DBMS không tin cậy cùng với một bộ lọc tin cậy.

Bảng 1 đưa ra một cái nhìn tổng quan về các kiến trúc được sử dụng trong một số DBMS thương mại và trong một số mẫu thử nghiên cứu của DBMS.

Bảng 1 Các kiến trúc mẫu thử DBMS và các sản phẩm thương mại

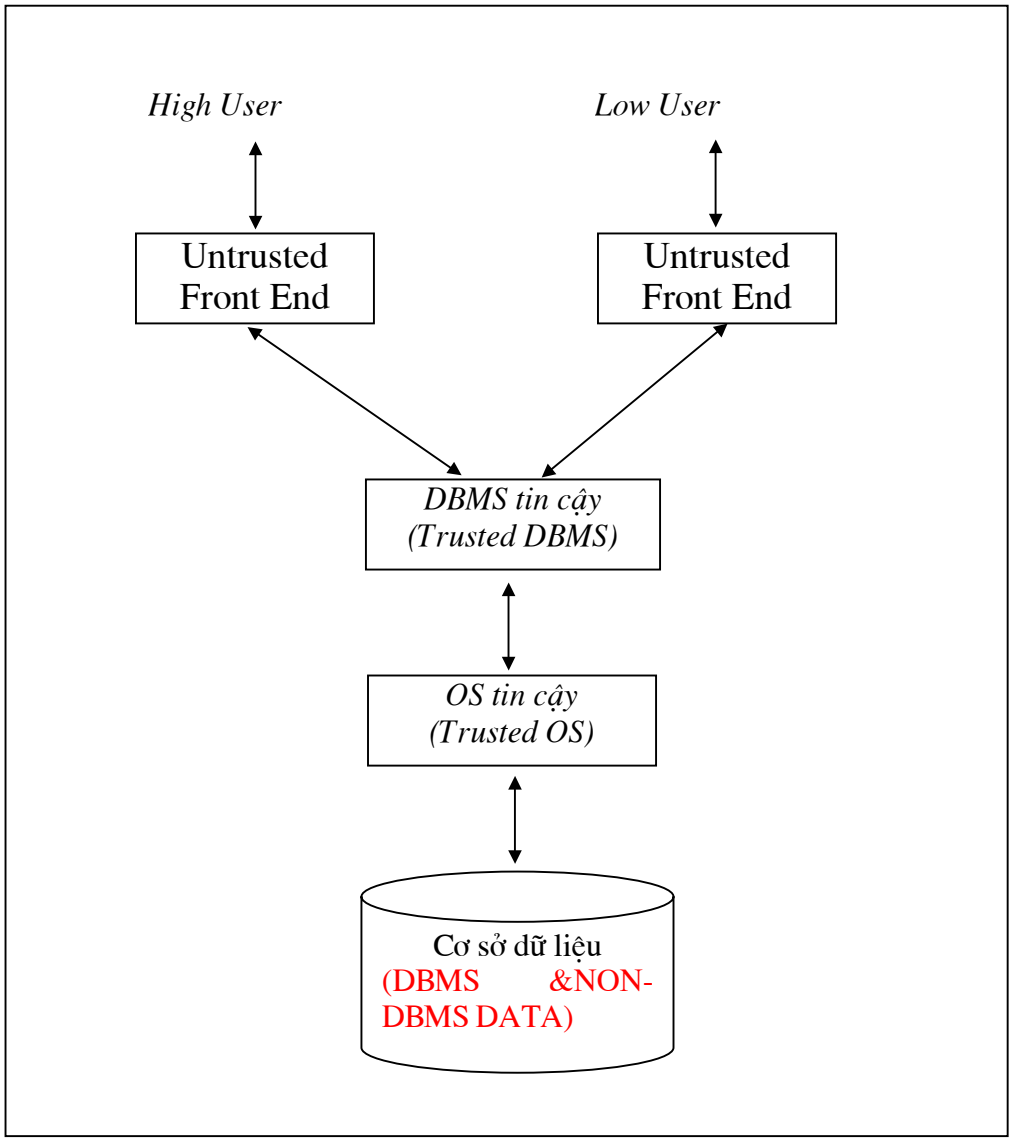
Kiến trúc	*Các mẫu thử nghiên cứu*	*DBMS thương mại*
Integrity Lock	Mitre	TRUDATA
Kernelized	Sea View	Oracle
Replicated	NRL	--------
Trusted Subject	A1 Secure DBMS (ASD)	Sybase
		Informix
		Ingres
		Oracle
		DEC
		Rubix

♣ Kiến trúc Trusted Subject (kiến trúc chủ thể tin cậy)

Kiến trúc chủ thể tin cậy được minh hoạ trong hình 1. Một tập hợp các UFE (*untrusted front end*) được sử dụng để tương tác với người sử dụng, với các mức cho phép khác nhau (Như đã được trình bày trong hình vẽ, có mức cao và mức thấp).

Khi một DBMS tin cậy được sử dụng và hoạt động như là một chủ thể tin cậy đối với OS, thì nó cũng được tin cậy, nó thực hiện các truy nhập vật lý vào cơ sở dữ liệu. Hoạt động như là một chủ thể tin cậy của OS có nghĩa là được miễn một hoặc nhiều khía cạnh nào đó trong chính sách an toàn của OS, nói chung, được miễn các kiểm soát bắt buộc.

DBMS và OS phải được nhìn nhận như là một thực thể, nếu hiểu theo nghĩa thông thường, chúng được ước tính để xác định mức bảo vệ. Trong kiến trúc này, DBMS có trách nhiệm trong việc bảo vệ đa mức các đối tượng của cơ sở dữ liệu.



Hình 1 Kiến trúc chủ thể tin cậy

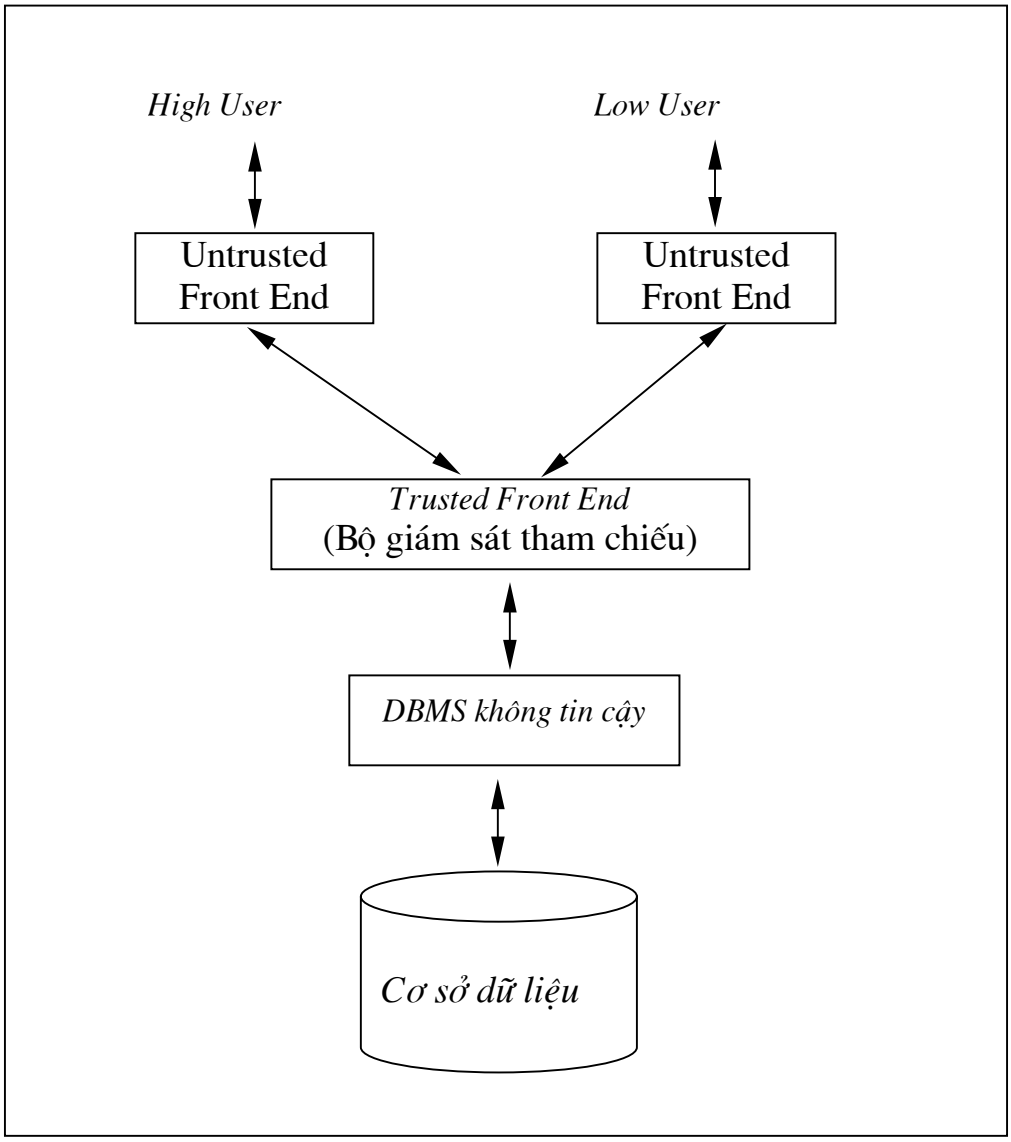
Lưới an toàn được xây dựng theo cách như vậy, định nghĩa mức High, mức Low và một mức DBMS không thích hợp với High và Low. Nhãn DBMS được gán cho cả các đối tượng và chủ thể. Chỉ có các chủ thể của DBMS có thể thực hiện các chương trình và truy nhập dữ liệu với một nhãn DBMS. Hơn nữa, các chủ thể (có nhãn DBMS) được xem như là các chủ thể tin cậy và được miễn các kiểm soát bắt buộc của OS. Theo giải pháp này, có thể nhóm các yếu tố có cùng mức nhạy cảm và lưu giữ chúng trong một đối tượng với mức chi tiết thô, chỉ gán một nhãn cho đối tượng này, hoặc gán nhãn cho từng đối tượng (ví dụ, các bộ, các giá trị). Sybase DBMS tuân theo giải pháp này, với kiến trúc máy khách/máy chủ. Sybase thực hiện gán nhãn mức bộ.

♣ Các kiến trúc Woods Hole

Các kiến trúc Woods Hole được phân loại như sau:

- Kiến trúc Integrity Lock
- Kiến trúc Kernelized
- Kiến trúc Replicated (còn được gọi là kiến trúc Distributed)

Chúng có thể được miêu tả thông qua một kiến trúc tổng quát, được minh hoạ trong hình 2.

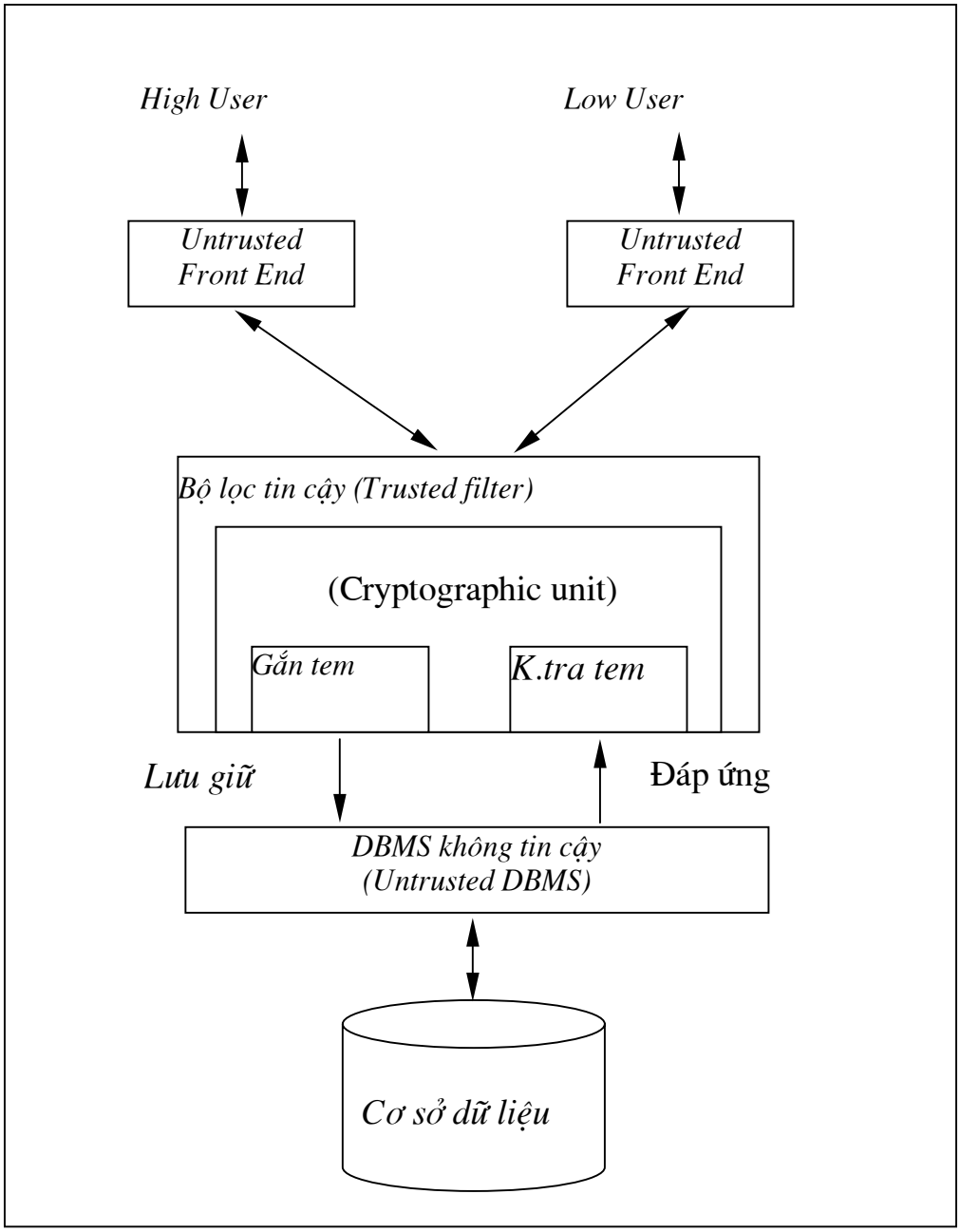


Hình 2 Các kiến trúc Woods Hole

Chúng ta nhận thấy rằng một tập hợp các UFE tương tác với những người sử dụng hoạt động tại các mức cho phép khác nhau, ở đây chúng được đơn giản hoá thành High và Low. Lần lượt, UFE tương tác với một TFE (*trusted front end*), nó hoạt động như một bộ giám sát tham chiếu; Có nghĩa là không thể bỏ qua nó. TFE tương tác với một UBED (*untrusted back end DBMS*), có trách nhiệm trong việc truy nhập dữ liệu vào cơ sở dữ liệu. Tiếp theo chúng ta mô tả các đặc điểm của từng kiến trúc.

- Kiến trúc Integrity Lock

Kiến trúc này được trình bày trong hình sau đây.



Hình 3 Kiến trúc Integrity Lock

Theo giải pháp này, người sử dụng được kết nối thông qua các giao diện front end không tin cậy, thực hiện tiền xử lý và hậu xử lý các câu truy vấn. Một TFE (còn được gọi là một bộ lọc tin cậy) được chèn vào giữa các UFE và DBMS không tin cậy. TFE có trách nhiệm trong việc tuân theo các chức năng an toàn và bảo vệ đa mức, hoạt động như là một TCB. TFE tuân theo bảo vệ đa mức bằng cách gắn nhãn an toàn cho các đối tượng của cơ sở dữ liệu, theo các dạng tem. Tem là một trường đặc biệt của một đối tượng, lưu giữ các thông tin (liên quan đến nhãn an

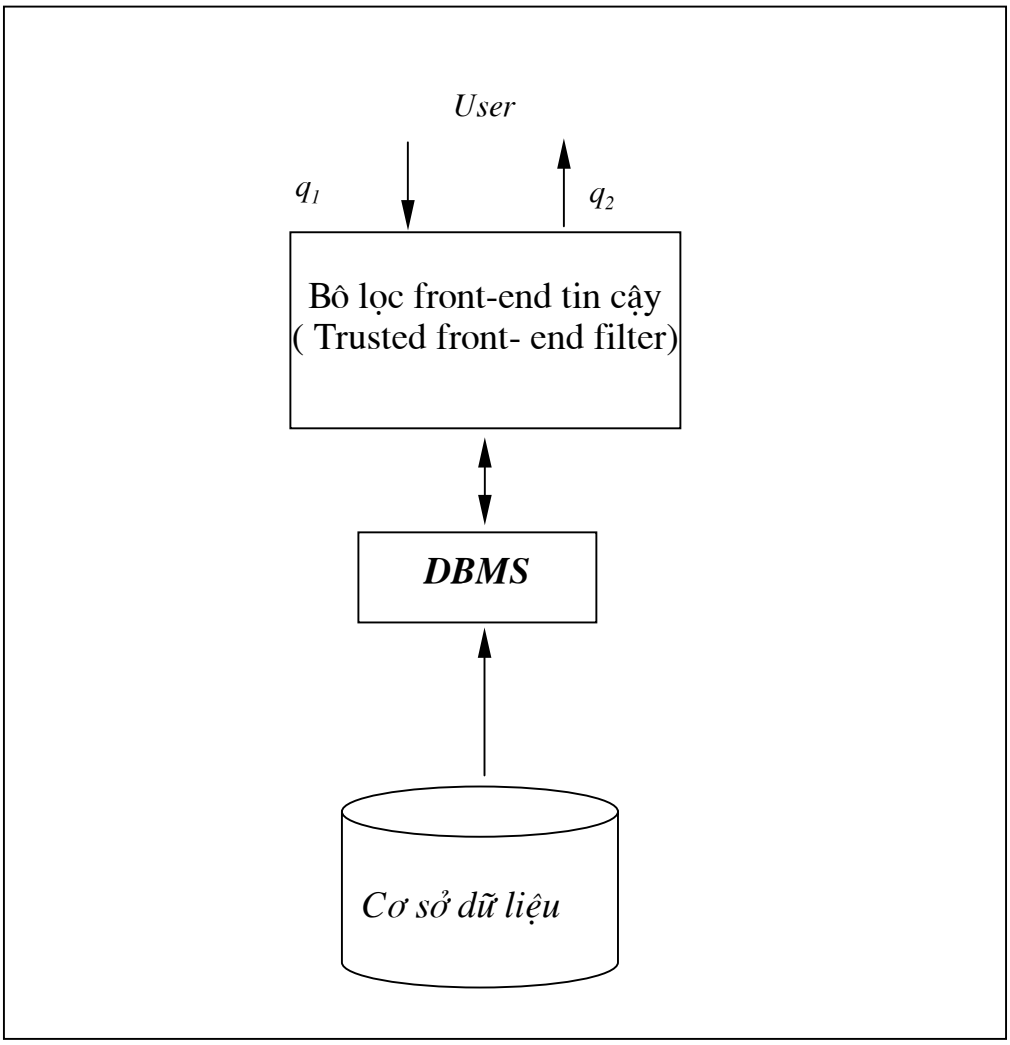
toàn và các dữ liệu kiểm soát liên quan khác) trong một khuôn dạng đã được mã hoá, được tạo ra bằng cách sử dụng một kỹ thuật niêm phong mật mã (*cryptoseal mechanism*), được gọi là Integrity Lock. TFE tiến hành tạo và phê chuẩn các tem, ngay khi dữ liệu được lưu giữ và nhận được từ cơ sở dữ liệu. TFE sinh ra các tem, bằng cách sử dụng các kỹ thuật tổng kiểm tra (*checksum*) (Nó sử dụng một hoặc nhiều khoá bí mật, chỉ duy nhất TFE biết được khoá này), bao quanh dữ liệu và được lưu giữ trong cơ sở dữ liệu theo một khuôn dạng đã được mã hoá. Tại thời điểm nhận lại, TFE tính toán lại các tem và so khớp với bản được lưu giữ, để phát hiện ra sự sai khớp, trước khi dữ liệu được chuyển cho người sử dụng. TFE có trách nhiệm tạo ra các bản ghi kiểm toán của riêng nó (có thể có cùng khuôn dạng với các bản ghi kiểm toán được OS tạo ra), để đảm bảo tính sẵn sàng của một vết kiểm toán thuần nhất.

Thậm chí, nếu tuân theo cơ chế dựa vào tem (*stamp-based machanism*) một cách chính xác, thì cũng chưa đủ đảm bảo an toàn. Trong thực tế, nó chỉ đảm bảo cho các trường hợp sau không xảy ra: truy nhập trực tiếp vào dữ liệu không được phép (hay là truy nhập trái phép dữ liệu), chuyển các thông tin không được phép vào các lớp phân loại không chính xác (thông qua con ngựa thành Tơroa).Với kiểu kiến trúc này, để tránh được các đe doạ trên, các phép chọn (*selections*), phép chiếu (*projections*), xử lý câu truy vấn phụ (*subquery handling*), tối ưu câu truy vấn (*query optimization*) và các phép thống kê (*statistical operations*) phải được cài vào trong TFE hoặc UFE, không được cài vào DBMS, DBMS chỉ có trách nhiệm đối với các phép toán lưu giữ và lấy lại. Theo cách này, TFE xem tất cả các dữ liệu (được yêu cầu) để trả lời câu truy vấn và được phép loại trừ khung nhìn dữ liệu (được trả lại), người sử dụng không được biết dữ liệu này.

Một giải pháp dành cho việc loại trừ các rủi ro suy diễn đã được đề xuất năm 1985, trong đó, một bộ lọc thay thế (*commutative filter*) được chèn vào giữa DBMS và người sử dụng, đảm bảo loại trừ được các đe doạ suy diễn, DBMS tránh được con ngựa thành Tơroa. Giải pháp này xuất phát từ giải pháp *Maximal Authorized View* (1977).

Theo Maximal Authorized View, mỗi câu truy vấn *q* được ước lượng dựa vào một khung nhìn của cơ sở dữ liệu, cơ sở dữ liệu này chỉ bao gồm dữ liệu mà người sử dụng đã biết (được gọi là khung nhìn được phép tối đa, nó là một tập hợp con

của dữ liệu được lưu giữ trong cơ sở dữ liệu), đưa ra nguồn gốc cho một câu truy vấn *qsec*, tránh suy diễn trên dữ liệu không được phép.



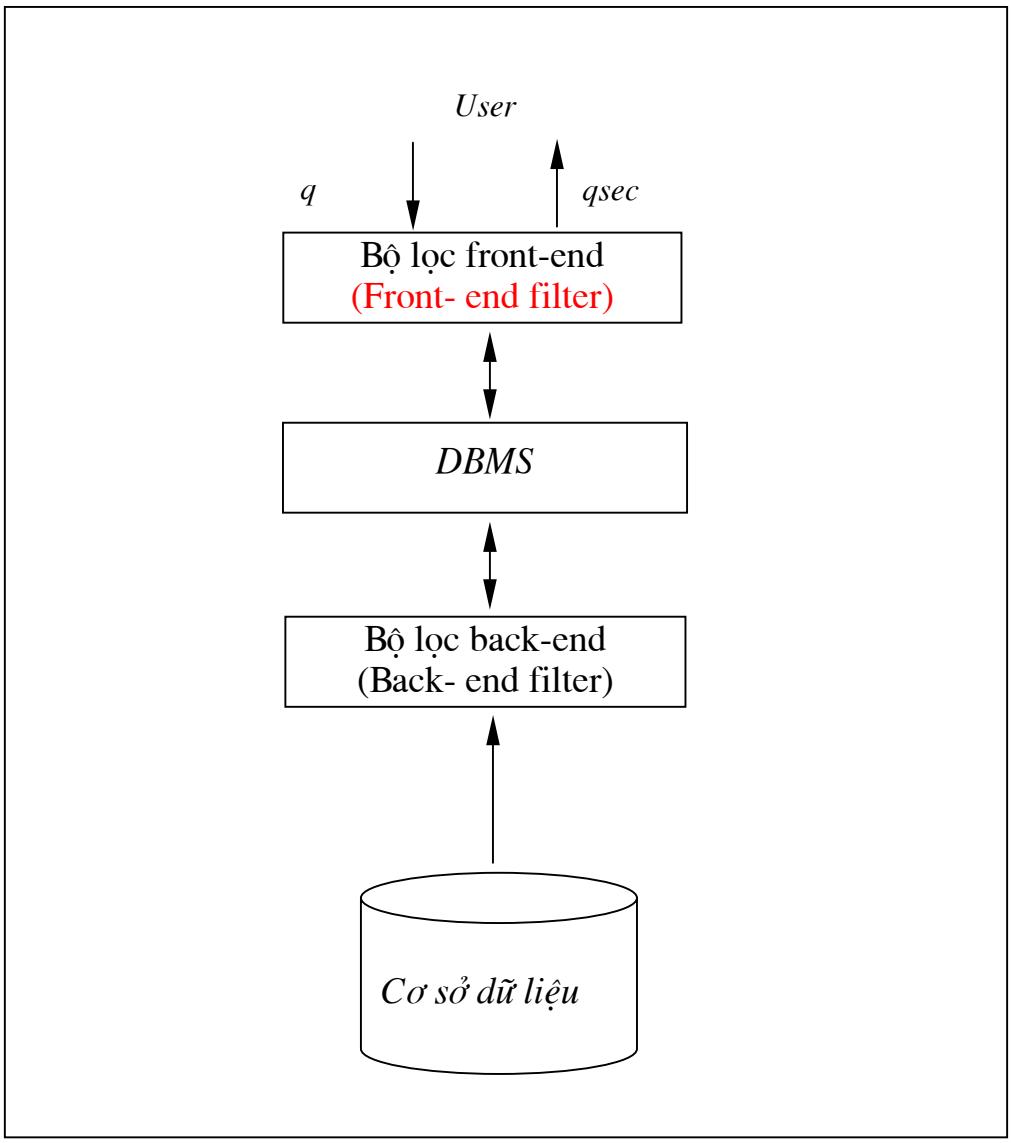
Hình 4 Giải pháp bộ lọc thay thế.

Bộ lọc back-end (còn được gọi là bộ lọc quản lý dữ liệu) có trách nhiệm trong việc định nghĩa khung nhìn được phép tối đa, bằng cách phát hiện tất cả các bản ghi/các thuộc tính không được phép, thay thế các yếu tố không được phép bằng giá trị 0.

Bộ lọc front-end của kiến trúc được trình bày trong hình 4. làm việc theo cách như vậy, câu truy vấn q_2 (được trả lại cho người sử dụng) tương đương với câu truy vấn *qsec* của kiến trúc trong hình 5., bằng cách bổ sung thêm cho câu truy vấn *q* (đây là câu truy vấn ban đầu của người sử dụng) thông tin về tem (đưa ra nguồn gốc cho câu truy vấn q_1) và lọc câu truy vấn q_2 từ đáp ứng của câu truy vấn q_1.

Bản gốc báo cáo không có trang 54

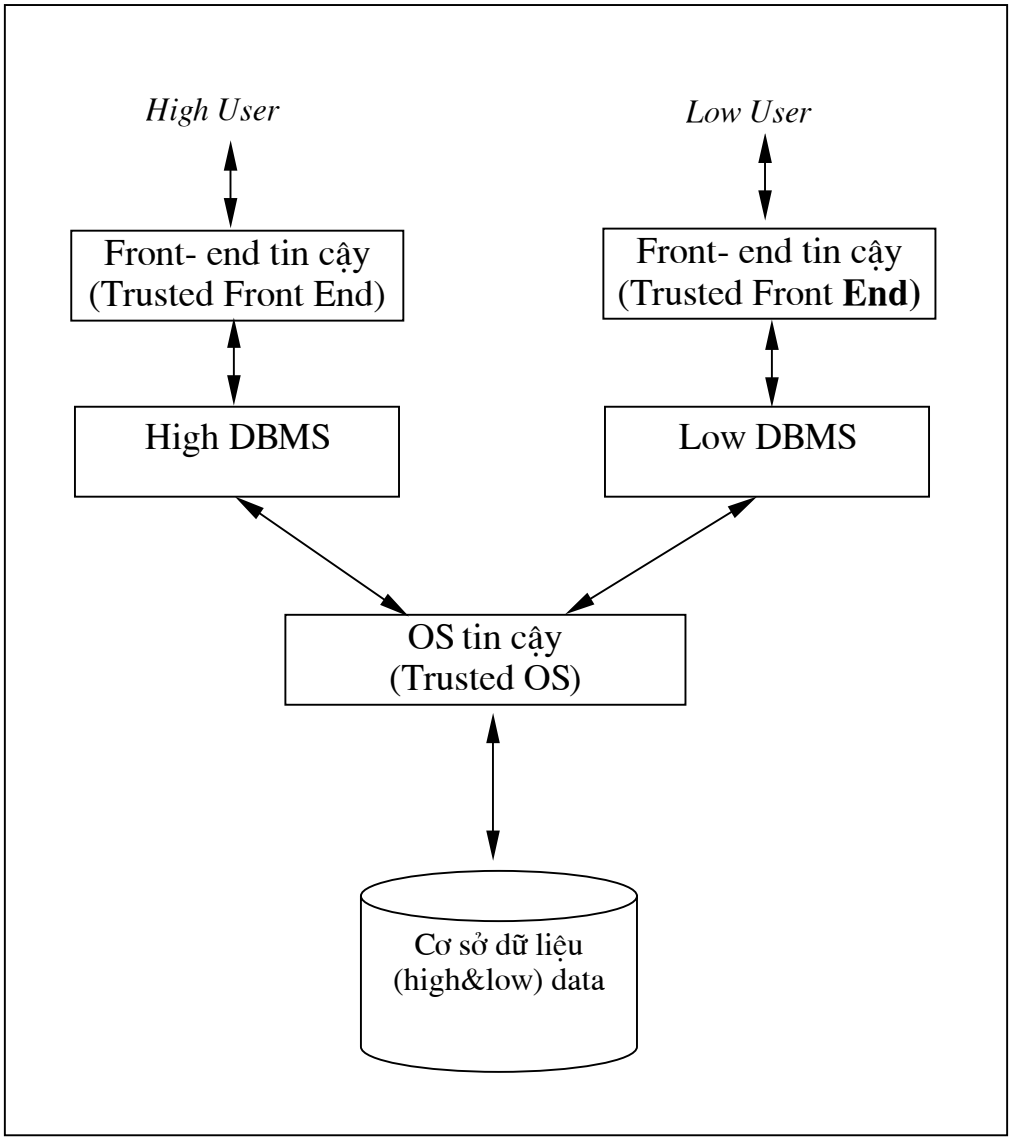
(Thông tin vẫn đầy đủ)



Hình 5 Giải pháp khung nhìn cho phép tối đa

- Kiến trúc Kernelized

Kiến trúc này được trình bày trong hình 6.



Hình 6 Kiến trúc Kernelized

Ở đây có sử dụng một OS tin cậy, nó có trách nhiệm đối với các truy nhập vật lý vào dữ liệu (trong cơ sở dữ liệu) và có trách nhiệm tuân theo bảo vệ bắt buộc. High User (người sử dụng làm việc ở mức cao) tương tác với một High DBMS, thông qua một TFE, Low User (người sử dụng làm việc ở mức thấp) tương tác với một Low DBMS. Sau đó, các yêu cầu của họ được chuyển cho OS, nó lấy lại dữ liệu hợp lệ từ cơ sở dữ liệu.

Theo giải pháp này, các đối tượng (có các nhãn an toàn giống nhau) của cơ sở dữ liệu được lưu giữ trong các đối tượng của OS tin cậy (đóng vai trò như là các kho chứa đối tượng của cơ sở dữ liệu). Vì vậy, OS tin cậy tiến hành kiểm soát an toàn trên các đối tượng này, cần có các quá trình phân tách và khôi phục quan hệ đa

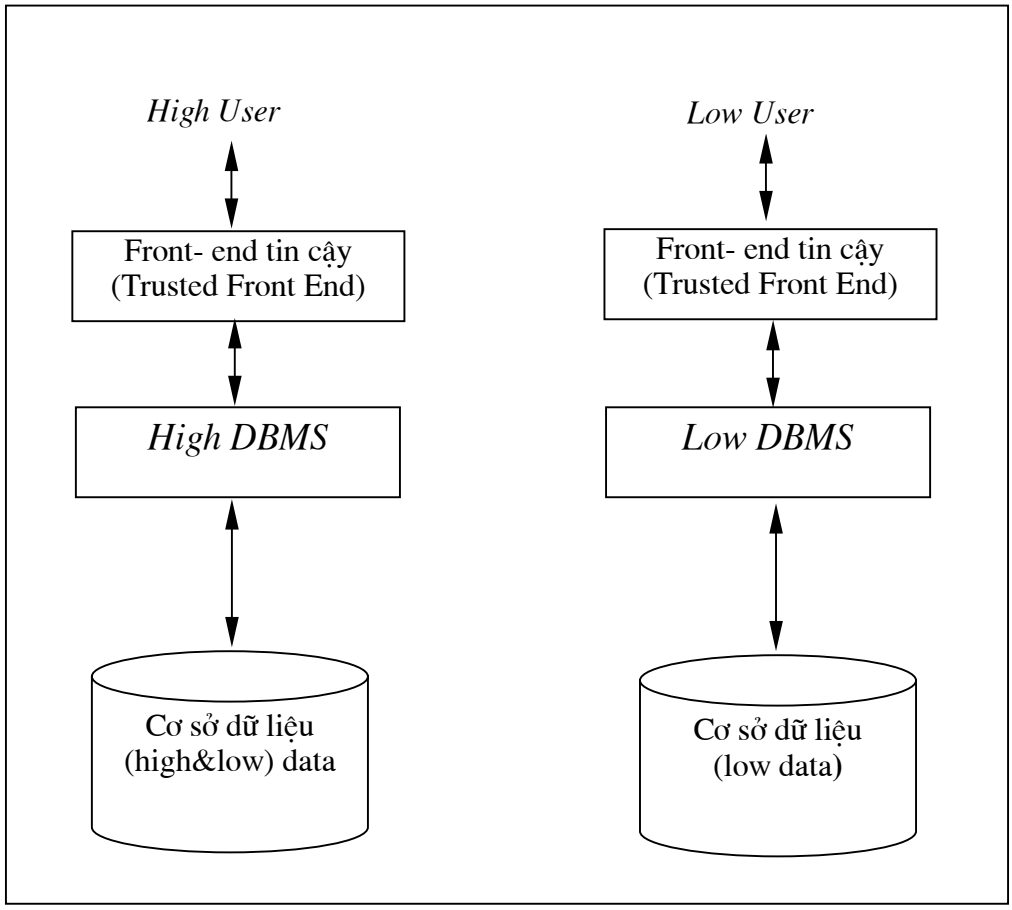
mức. Quá trình phân tách được thực hiện khi chuyển đổi một quan hệ đa mức thành một số quan hệ đơn mức, khi chỉ chứa dữ liệu ở một mức an toàn xác định nào đó, chúng được lưu giữ trong các đối tượng của hệ điều hành. Quá trình khôi phục được thực hiện trên các quan hệ đơn mức khi chúng được lấy lại, nhằm sinh ra một khung nhìn đa mức chỉ chứa các dữ liệu mà người sử dụng (người yêu cầu câu truy vấn) đã biết. Các thuật toán phân tách và khôi phục phải được định nghĩa chính xác, nhằm đảm bảo tính đúng đắn và hiệu quả của hệ thống.

Các bản ghi kiểm toán (được OS tin cậy sinh ra cho các phép toán liên quan đến truy nhập vào các đối tượng của OS) và các bản ghi kiểm toán khác phải được sinh ra cho các phép toán của DBMS và chúng được ghi lại trong một vết kiểm toán mức hệ thống cao, có thể có cùng khuôn dạng với các bản ghi kiểm toán của OS. Kiến trúc này được sử dụng trong mẫu thử nghiên cứu Sea View và DBMS Oracle thương mại.

- Kiến trúc Replicated (lặp)

Kiến trúc này được trình bày trong hình 7.

Theo giải pháp này, dữ liệu mức thấp được lặp trong cơ sở dữ liệu. Theo cách này, người dùng mức thấp chỉ được phép truy nhập vào cơ sở dữ liệu độ ưu tiên thấp, không có khả năng sửa đổi dữ liệu mức cao. Để tuân theo giải pháp này cần có các thuật toán đồng bộ an toàn để đảm bảo tính tương thích lặp và chi phí (do lặp) tăng dần theo kích cỡ của lưới an toàn. Không một DBMS thương mại nào sử dụng kiến trúc này vì nó rất đắt, do phải lặp dữ liệu; Nó chỉ được sử dụng trong mẫu thử nghiên cứu NRL.



Hình 7 Kiến trúc Replicated

- Nhận xét về các kiến trúc an toàn

Các kiến trúc an toàn được trình bày ở trên thích hợp cho các mục đích khác nhau, tuỳ thuộc vào các đặc điểm và các yêu cầu của miền ứng dụng đích. Ví dụ, kiến trúc Kernelized phù hợp với các môi trường có yêu cầu bảng đơn mức, bởi vì nó kinh tế nhất và dễ thực hiện nhất. Đối với những môi trường mà DBMS đã định rõ đặc điểm yêu cầu nhãn mềm dẻo và một mức tích hợp cao giữa DBMS và OS cơ sở, kiến trúc Integrity Lock phù hợp hơn cả. Kiến trúc chủ thể tin cậy thích hợp với các miền ứng dụng (đây là nơi có thể đảm bảo một đường dẫn tin cậy từ các ứng dụng đến DBMS).

Khi đánh giá mức tin cậy của các kiến trúc, lưu ý rằng độ phức tạp trong vấn đề đánh giá phụ thuộc vào kiến trúc. Ví dụ, kiến trúc Integrity Lock được phê chuẩn một cách dễ dàng nhất, trong khi đó kiến trúc chủ thể tin cậy thì phức tạp hơn. Thực vậy, trong khi chỉ với một bộ lọc kích cỡ nhỏ, bao gồm các dịch vụ thông

thường do một OS tin cậy cung cấp, chúng ta lại phải đánh giá một DBMS tin cậy. Kiến trúc Kernelized nằm ở vị trí trung gian, nhưng nếu phải bổ sung thêm phần mềm tin cậy nhằm đảm bảo hoạt động an toàn trong một môi trường đa mức, thì việc đánh giá trở nên khó khăn hơn.

Còn một vấn đề khác liên quan đến mức độ phụ thuộc giữa DBMS và OS cơ sở tin cậy. Các kiến trúc Integrity Lock và Kernelized dựa vào các dịch vụ an toàn do OS cơ sở tin cậy cung cấp, trong khi đó kiến trúc chủ thể tin cậy đưa ra một mức phụ thuộc và tích hợp thấp hơn. Khi gán độ chi tiết, có nghĩa là đối tượng nhỏ nhất của cơ sở dữ liệu có thể được gán một nhãn. Các kiến trúc tiến hành gán khác nhau.

Ví dụ, kiến trúc Integrity Lock và kiến trúc thực thể tin cậy cung cấp khả năng gán nhãn hàng, trong khi đó việc gán nhãn của kiến trúc Kernelized do OS cung cấp, trên các đối tượng có trong kho chứa của nó, vì vậy giảm tổng chi phí lưu giữ. Tuy nhiên, cơ chế gán nhãn sau sẽ không thích hợp nếu cần phải quản lý các bảng đa mức. Hơn nữa, kiến trúc Integrity Lock và kiến trúc chủ thể tin cậy có thể được mở rộng chính đáng, nhằm hỗ trợ cho việc gán nhãn tại mức trường của một hàng, trong khi đó kiến trúc Kernelized lại không cần.

2. 4 Thiết kế các cơ sở dữ liệu an toàn

An toàn cơ sở dữ liệu có thể được nhìn nhận như là một yêu cầu thứ hai (được bổ sung thêm vào các hệ thống hiện có) hoặc được coi như là một đòi hỏi chủ yếu. Chính vì vậy, nó được coi là một yêu cầu thích đáng trong các giai đoạn thiết kế hệ thống ban đầu.

Trong hầu hết các trường hợp, an toàn không phải là một mối quan tâm chủ yếu trong việc phát triển hệ thống. Nhờ đó, các hệ thống sẽ trở nên phong phú thêm với các gói an toàn, đưa ra các đặc trưng an toàn cơ bản mức OS (xác thực người dùng, kiểm soát truy nhập, kiểm toán). Điều này đã xảy ra đối với nhiều OS được sử dụng rộng rãi, chẳng hạn như MVS, VMS và VM, an toàn được hỗ trợ thông qua các gói RACF, Top Secret và CA-ACF2.

Trong một số môi trường (ví dụ, trong môi trường quân sự), hệ thống an toàn cần được định nghĩa một cách phi thể thức và các yêu cầu bảo vệ được kiểm tra một cách hình thức. Trong bất kỳ trường hợp nào, khi thiết kế các hệ thống an toàn cơ sở dữ liệu, chúng ta phải đối mặt với rất nhiều vấn đề trọng yếu và nhiều vấn đề nghiên cứu còn bị bỏ ngỏ.

Tiêu chuẩn DoD bao gồm các chuẩn tham chiếu hữu ích cho việc phân loại hệ thống phần mềm an toàn, đồng thời cung cấp hướng dẫn cho việc thiết kế an toàn. Trong thực tế, tại mỗi mức phân loại, với một tập hợp các yêu cầu đã được mô tả, nếu chúng được quan tâm trong quá trình thiết kế hệ thống thì có thể đảm bảo được hiệu suất mong muốn. Nói riêng, tiêu chuẩn DoD trình bày một cách rõ ràng một tập hợp các yêu cầu thiết yếu nên được thực hiện khi thiết kế hệ thống, liên quan đến việc định nghĩa mô hình khái niệm của các yêu cầu bảo vệ hệ thống và các chính sách an toàn hệ thống (bắt buộc và tuỳ ý), chúng có thể được sửa đổi và kiểm tra thử nghiệm, bằng cách sử dụng các kỹ thuật kiểm tra hình thức. Hơn nữa, rõ ràng là tiêu chuẩn DoD liên quan tới một TCB, nó được sử dụng để tuân theo các chính sách và dàn xếp tất cả các truy nhập vào dữ liệu.

Từ các mối quan tâm trên, chúng ta có thể đảm bảo an toàn bằng cách xác định rõ các yêu cầu bảo vệ của một hệ thống và sau đó thực hiện các cơ chế an toàn có sử dụng các phương pháp và các kỹ thuật đã được trang bị.

Một hướng tiếp cận mang tính phương pháp luận (trong đó tham chiếu rõ ràng vào các yêu cầu của DoD) có thể là một câu trả lời cho vấn đề thiết kế cơ sở dữ liệu an toàn với các đặc tính an toàn, thông qua các giai đoạn phát triển ban đầu.

Một phương pháp luận đa giai đoạn trình bày một hướng tiếp cận thích hợp cho việc thiết kế cơ sở dữ liệu an toàn, cho phép các nhà thiết kế xác định một cách chính xác các yêu cầu an toàn của một môi trường.

Trong thực tế, việc tiếp cận thiết kế cơ sở dữ liệu an toàn bắt đầu từ các chức năng an toàn (do OS và DBMS đưa ra) là không thoả đáng, mặc dù các gói và các sản phẩm an toàn đã có sẵn và có thể được xem xét đến. Tương tự, ngày nay không ai muốn thiết kế một cơ sở dữ liệu bắt đầu từ một DBMS xác định.

Hơn nữa, chúng ta đã đưa ra các mô hình, các cơ chế và các gói hướng tập trung vào các vấn đề an toàn. Mô hình là một cách hình thức hoá việc miêu tả các yêu cầu và các chính sách an toàn của hệ thống; Các cơ chế của OS cung cấp các chức năng an toàn cơ bản (ví dụ: nhận dạng/xác thực, kiểm soát truy nhập); Cuối cùng, các gói và các DBMS an toàn đã mở rộng chức năng của OS, nhằm quản lý các yêu cầu an toàn của cơ sở dữ liệu. Các mô hình, cơ chế và sản phẩm an toàn hình thành một phương pháp luận tích hợp đa giai đoạn (*integrated multiphase methodology*), hỗ trợ phát triển (một cách có hệ thống) các hệ thống cơ sở dữ liệu an toàn thông qua các giai đoạn phân tích và thiết kế ban đầu. Nói riêng, phương pháp luận hướng

dẫn các nhà phát triển trong quá trình phân tích các yêu cầu an toàn, lựa chọn các chính sách an toàn, định nghĩa một mô hình an toàn và thiết kế các cơ chế an toàn để thực hiện mô hình, quan tâm đến các tính năng an toàn hiện tại của OS và DBMS. Phương pháp luận (chúng ta đề xuất khi thiết kế cơ sở dữ liệu an toàn) dựa trên các nguyên tắc (do tiêu chuẩn DoD đưa ra), bao gồm các giai đoạn sau:

(1) Phân tích sơ bộ

(2) Các yêu cầu và các chính sách an toàn

(3) Thiết kế khái niệm

(4) Thiết kế lôgíc

(5) Thiết kế vật lý

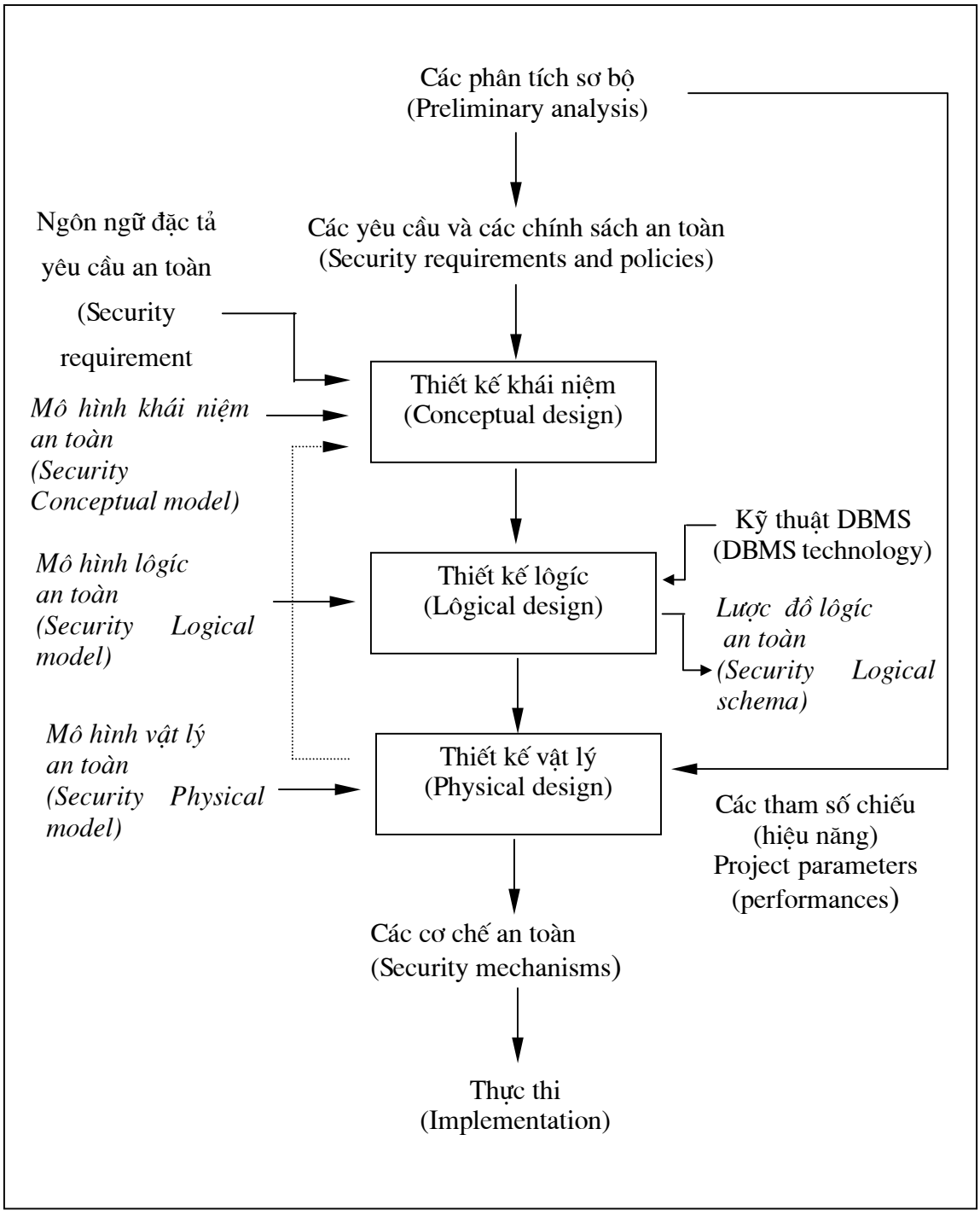
Tất cả được trình bày trong hình 8.

Phương pháp luận phát triển đa giai đoạn mang lại rất nhiều lợi ích.

Trước hết, nó có thể chia nhỏ quá trình thiết kế (nói chung, đây là một nhiệm vụ phức tạp) thành các nhiệm vụ nhỏ hơn, vì vậy, nó cho phép các nhà phát triển tập trung vào các khía cạnh an toàn riêng của từng nhiệm vụ.

Hơn nữa, một hướng tiếp cận mang tính phương pháp luận tách chính sách an toàn ra khỏi các cơ chế an toàn. Chính sách là các nguyên tắc ở mức cao, bắt buộc phải tuân theo trong các quá trình thiết kế, thực thi và quản lý hệ thống an toàn. Chúng đưa ra các yêu cầu bảo vệ và các chiến lược có thể có khi bảo vệ thông tin của các tổ chức. Hiện có rất nhiều các chính sách kiểm soát an toàn khác nhau, chúng đưa ra các đòi hỏi/các chiến lược bảo vệ khác (không mâu thuẫn lẫn nhau), thích hợp với các môi trường khác nhau.

Cơ chế an toàn là một tập hợp các chức năng phần cứng, phần sụn và phần mềm tuân theo các chính sách. Các cơ chế nên được kiểm tra dựa trên các yêu cầu an toàn, để chứng minh rằng chúng thực sự tuân theo các đặc tả của chính sách xác định nào đó. Hơn nữa, các cơ chế nên có khả năng tuân theo một số chính sách.



Hình 8 Phương pháp luận thiết kế CSDL an toàn

Khi phát triển hệ thống an toàn, việc tách bạch các chính sách và các cơ chế mang lại một số thuận lợi sau:

- Khả năng định nghĩa các nguyên tắc kiểm soát truy nhập;

- Khả năng so sánh các chính sách kiểm soát truy nhập khác nhau; hoặc so sánh các cơ chế khác nhau nhưng dành cho cùng một chính sách;

- Khả năng định nghĩa các cơ chế hỗ trợ các chính sách khác nhau; Thuận lợi này trở thành một đòi hỏi quan trọng khi các chính sách thay đổi do các yêu cầu của tổ chức thay đổi.

Thứ hai, thuận lợi của phương pháp luận đa giai đoạn (dựa vào mô hình an toàn khái niệm) là nó có thể chứng minh được tính an toàn trong quá trình thiết kế hệ thống. Tính an toàn hệ thống có thể được chứng minh, bằng cách chứng minh tính đúng đắn của mô hình an toàn dựa vào các yêu cầu và chứng minh tính đúng đắn của các đặc tả trong cơ chế dựa vào mô hình an toàn.

Mô hình an toàn là:

- Một công cụ thiết kế nhằm hướng dẫn thiết kế hệ thống;

- Một cấu trúc dành cho nghiên cứu; Chúng ta có thể nghiên cứu hoặc so sánh các mô hình khái niệm khác nhau, không phụ thuộc vào các chi tiết thực thi. Chúng ta có thể chọn lựa các mô hình khái niệm khác nhau, sao cho phù hợp với thiết kế hiện tại; Hoặc như một sự lựa chọn, một mô hình mới có thể được phát triển từ việc trộn ghép, phi hình thức, sao cho phù hợp với các yêu cầu của hệ thống;

- Một công cụ mang tính giáo khoa nhằm đơn giản hoá việc miêu tả hệ thống;

- Một công cụ so sánh/đánh giá cho các hệ thống an toàn khác nhau.

Các giai đoạn của phương pháp luận được trình bày trong các mục nhỏ sau đây.

2.4.1. Giai đoạn phân tích sơ bộ

Mục đích của giai đoạn này là tiến hành nghiên cứu hệ thống an toàn (sau khi đã quyết định một mức chiến lược), chẳng hạn như phân tích những gì có thể xảy ra trong khi phát triển các hệ thống thông tin.

Việc nghiên cứu mang tính khả thi này bao gồm: đánh giá các rủi ro, ước lượng các chi phí thiết kế, phát triển các ứng dụng xác định nào và xác định quyền ưu tiên của chúng. Để đảm bảo được mục đích này, các phân tích cần quan tâm đến:

- Các rủi ro hệ thống (*system risks*): Đây là các đe doạ đáng kể nhất có thể xảy ra đối với một cơ sở dữ liệu, cần ước tính (đánh giá) các hình thức xâm phạm tương ứng và hậu quả của việc mất mát, thông qua các kỹ thuật phân tích rủi ro. Nói chung, các đe doạ điển hình (xảy ra trong các môi trường ứng dụng hoặc các tổ chức) là: đọc và sửa đổi trái phép dữ liệu, không cho phép (từ chối) người sử dụng hợp pháp truy nhập. Các hình thức tấn công phụ thuộc vào kiểu đe doạ. Ví dụ, chúng ta có thể phát hiện được việc đọc và sửa đổi trái phép dữ liệu, thông qua việc truy nhập vào khu vực lưu giữ vật lý, hoặc thông qua việc sử dụng không đúng đắn của các chương trình ứng dụng (ví dụ: con ngựa thành Tơroa), hoặc thông qua việc truy nhập vào các lược đồ dữ liệu.

- Các đặc trưng của môi trường cơ sở dữ liệu (*features of database environment*): Chúng ảnh hưởng đến các yêu cầu bảo vệ và các cơ chế liên quan. Ví dụ, các hệ thống bảo vệ đa mức phù hợp với môi trường quân sự, nhưng chưa chắc đã phù hợp với các môi trường thương mại, vì khó có thể định nghĩa các mức an toàn dữ liệu/người sử dụng trong đó.

- Khả năng ứng dụng của các sản phẩm an toàn hiện có (*applicability of existing security products*): Cần phải quan tâm đến tính tiện lợi khi chọn lựa giữa các sản phẩm thương mại hiện có sẵn và việc phát triển một hệ thống an toàn từ trộn ghép. Sự lựa chọn phụ thuộc vào loại hình, mức bảo vệ và khi nào thì an toàn được coi như là một đặc trưng vốn có của cơ sở dữ liệu; hay là một đặc trưng bổ sung.

- Khả năng tích hợp của các sản phẩm an toàn (*Integrability of the security products*): Đưa ra khả năng tích hợp các cơ chế an toàn với các cơ chế phần cứng và phần mềm thực tế.

- Hiệu năng đạt được của các hệ thống an toàn (*performance of the resulting security system*): Hiệu năng của các hệ thống an toàn cần được so sánh với các hệ thống thực tế, hoặc với các hệ thống mới, mà không cần bất kỳ các cơ chế và các kiểm soát an toàn nào.

Kết quả của các phân tích này là một tập hợp các đe doạ dễ xảy ra với một hệ thống, được sắp xếp theo quyền ưu tiên. Hơn nữa, chúng ta cần đánh giá khả năng

áp dụng và tích hợp của các sản phẩm thương mại an toàn với các cơ chế hiện tại, có thể phát triển các cơ chế phi hình thức theo yêu cầu thông qua việc trộn ghép.

Giai đoạn tiếp theo của phương pháp luận được thực hiện hay không còn tuỳ thuộc vào các phân tích chi phí/lợi ích có mang lại một kết quả khả quan hay không.

2.4.2 Phân tích yêu cầu và chọn lựa chính sách an toàn

Việc phân tích các yêu cầu an toàn bắt đầu từ việc nghiên cứu đầy đủ và chính xác tất cả các đe doạ có thể xảy ra với một hệ thống. Điều này cho phép các nhà thiết kế xác định các yêu cầu an toàn một cách chính xác và đầy đủ, tuỳ thuộc vào các đòi hỏi bảo vệ thực tế của hệ thống.

Các cơ sở dữ liệu khác nhau có các đòi hỏi bảo vệ khác nhau (cho các kiểu rủi ro khác nhau). Sự khác nhau đầu tiên xuất phát từ tính nhạy cảm của thông tin, tiếp đến là các luật và dự luật có thể có. Hơn nữa, các hệ thống được phân loại thành hệ thống rủi ro cao hoặc hệ thống rủi ro thấp, dựa vào các yếu tố cơ bản, chẳng hạn như mức tương quan dữ liệu, chia sẻ dữ liệu, khả năng truy nhập dữ liệu, kỹ năng của nhân viên và các kỹ thuật được chọn lựa.

Tính tương quan (*correlation*) xảy ra khi dữ liệu liên quan lẫn nhau, mọi sự thay đổi trên một mục dữ liệu kéo theo sự thay đổi của các mục dữ liệu khác. Tính tương quan cũng xảy ra khi thực hiện các phép đọc trên một mục dữ liệu, vì đồng thời chúng ta cũng biết được giá trị của các mục dữ liệu khác.

Chia sẻ dữ liệu xảy ra khi nhiều ứng dụng (hoặc nhiều người sử dụng) sử dụng chung một mục dữ liệu. Nói riêng trong cơ sở dữ liệu, điều này dẫn đến các vấn đề tương tranh (*concurrency problems*), nguyên nhân là do có nhiều truy nhập đồng thời vào dữ liệu. Chia sẻ dữ liệu cũng dẫn đến các vấn đề về tính toàn vẹn và tính bí mật dữ liệu. Đối với tính bí mật, nhiều tiến trình (truy nhập vào cùng một mục dữ liệu) có thể biết được các thông tin do các tiến trình khác đã chèn vào trước đó, ảnh hưởng đến tính toàn vẹn dữ liệu, do các sửa đổi dữ liệu không chính xác (chủ tâm hoặc vô ý), điều này ảnh hưởng đến tính đúng đắn của các tiến trình khác khi chúng sử dụng dữ liệu.

Tóm lại, các hệ thống rủi ro thấp là các hệ thống ở đó dữ liệu được phân hoạch cao và các tiến trình phi tập trung, tránh được tình trạng tương quan và chia sẻ. Thậm chí còn đưa ra độ dư thừa dữ liệu nào đó.

Khả năng truy nhập dữ liệu liên quan đến vấn đề ai là người có thể truy nhập vào dữ liệu này với các mục đích nào đó. Chúng ta phải xác định khả năng truy nhập vào các kiểu dữ liệu khác nhau, quan tâm đến các khía cạnh khác nhau, chẳng hạn như tính riêng tư, tính bí mật, tính tin cậy, các quyền và các quan hệ hợp pháp. Thêm vào đó, người sử dụng (người yêu cầu truy nhập vào dữ liệu) nên được trao các quyền theo cách như này, nhằm trợ giúp cho việc phân định trách nhiệm. Về khả năng truy nhập, các hệ thống (nơi áp dụng các chính sách liên quan đến tính riêng tư, bí mật và phân định trách nhiệm) chứng minh là các hệ thống rủi ro thấp. Các rủi ro tăng dần trong các hệ thống có truy nhập trong thời gian thực, bởi vì tất cả người sử dụng có thể truy nhập vào toàn bộ dữ liệu.

Số lượng và kiểu người sử dụng cũng ảnh hưởng đến việc bảo vệ hệ thống. Về một khía cạnh nào đó, các tấn công an toàn trở nên thường xuyên hơn khi người sử dụng lạm dụng các quyền của họ.

Hơn nữa, an toàn phụ thuộc vào các kỹ thuật được chọn lựa. Các hệ thống rủi ro thấp sử dụng phần cứng và phần mềm đã được chứng nhận, từ các nhà cung cấp được chứng nhận, hoặc có thể sử dụng các hệ thống thử nghiệm rộng rãi, do một số quốc gia chưa có cơ quan chứng thực.

Trong quá trình phân tích yêu cầu, chúng ta cần xác định được các điểm yếu dễ bị tấn công của hệ thống, bằng cách quan tâm đến các tấn công dễ xảy ra nhất (các tấn công do chủ ý/vô ý). Các tấn công phổ biến nhất là khám phá và sửa đổi trái phép dữ liệu, hoặc từ chối truy nhập dữ liệu.

Một hướng tiếp cận mang tính hệ thống được sử dụng khi phân tích đe doạ và các điểm yếu dễ bị tấn công của hệ thống, như sau:

- Phân tích giá trị (*value analysis*): Phân tích dữ liệu được lưu giữ và các ứng dụng truy nhập vào dữ liệu này, nhằm xác định mức nhạy cảm của chúng. Các kiểm soát truy nhập tăng theo mức nhạy cảm của dữ liệu.

- Nhận dạng đe doạ (*threat identification*): Cần nhận dạng các đe doạ điển hình, cũng như các kỹ thuật xâm nhập (có thể có) của các ứng dụng khác nhau.

- Phân tích các điểm yếu dễ bị tấn công (*vulnerability analysis*): Cần nhận dạng các điểm yếu của hệ thống và liên hệ chúng với các đe doạ đã được nhận dạng từ trước.

- Phân tích rủi ro (*risk analysis*): Đánh giá các đe doạ, các điểm yếu của hệ thống và các kỹ thuật xâm nhập, dựa vào các xâm phạm tính bí mật, tính toàn vẹn của hệ thống (chẳng hạn như : khám phá, xử lý trái phép dữ liệu, sử dụng trái phép tài nguyên và từ chối dịch vụ).

- Ước tính rủi ro (*risk evaluation*): Cần ước tính khả năng xảy ra của từng biến cố không mong muốn, kết hợp với khả năng phản ứng hoặc đối phó của hệ thống đối với các biến cố này.

- Xác định yêu cầu (*requirement definition*): Cần xác định các yêu cầu an toàn, dựa vào các đe doạ (đã được ước tính) và các biến cố không mong muốn, đồng thời dựa vào khả năng xuất hiện của chúng.

Cần xác định rõ các chính sách bảo vệ, dựa vào các yêu cầu bảo vệ đã được nhận dạng. Chúng ta cần thực hiện giai đoạn này để xác định các yêu cầu bảo vệ cho cơ sở dữ liệu, có nghĩa là định nghĩa các chế độ truy nhập của chủ thể vào các đối tượng của hệ thống. Cần định nghĩa các lớp người sử dụng, mỗi lớp có các quyền truy nhập xác định vào cơ sở dữ liệu. Chính vì vậy, chúng ta có thể liệt kê được một tập hợp các câu phi hình thức - nêu rõ các yêu cầu an toàn, chẳng hạn như : "Quyền được tập trung"; "Các giao tác cập nhật lương phải do một thiết bị đầu cuối thực hiện"; "Chỉ có nhân viên thị trường được phép truy nhập vào thông tin thị trường". Việc lựa chọn các chính sách tuân theo các yêu cầu đã được xác định.

➢ Việc lựa chọn các chính sách an toàn

Mục đích của một chính sách an toàn là: định nghĩa các quyền truy nhập hợp pháp của mỗi chủ thể vào các đối tượng khác nhau trong hệ thống, thông qua một bộ các nguyên tắc.

Các chính sách an toàn phù hợp với các yêu cầu an toàn xác định đã được chọn lựa, định nghĩa chi tiết các chế độ truy nhập (ví dụ: đọc, ghi) mà mỗi chủ thể (hoặc nhóm) sử dụng trên mỗi đối tượng (hoặc một tập hợp các đối tượng).

Các chính sách an toàn cơ bản có thể được kết hợp với nhau, nhằm đáp ứng tốt hơn các yêu cầu an toàn.

Để hỗ trợ cho việc chọn lựa các chính sách an toàn, một bộ tiêu chuẩn chọn lựa chính sách gồm có:

- Tính bí mật đối nghịch tính toàn vẹn đối nghịch tính tin cậy (*secrecy versus integrity versus reliability*): Tính bí mật là quan trọng nhất, ví dụ trong môi trường quân sự; Tính toàn vẹn và tính tin cậy trong môi trường thương mại.

- Chia sẻ tối đa đối nghịch đặc quyền tối thiểu (*maximum sharing versus minimum privilege*): Tuỳ thuộc vào đặc quyền tối thiểu (cần-để-biết), người sử dụng chỉ được phép truy nhập vào các thông tin cần thiết tối thiểu cho nhiệm vụ của họ; Điều này thích hợp cho môi trường quân sự. Tuy nhiên, các môi trường (giống như các trung tâm nghiên cứu, hoặc các trường đại học) nên có một chính sách chia sẻ tối thiểu.

- Mức độ chi tiết của kiểm soát (*granularity of control*): Thứ nhất, khi nói đến mức độ chi tiết của kiểm soát, người ta muốn nói đến phạm vi của các kiểm soát, nó liên quan đến số lượng các chủ thể và các đối tượng bị kiểm soát. Chúng ta có thể đạt được kiểm soát toàn cục (trên tất cả các thực thể của hệ thống) thông qua các chính sách bắt buộc (*mandatory policies*); Kiểm soát từng phần (chỉ trên một số thực thể của hệ thống) được cung cấp thông qua các chính sách tùy ý (*discretionary policies*). Thứ hai, khi nói đến mức độ chi tiết của kiểm soát, người ta muốn nói đến độ chi tiết của các đối tượng bị kiểm soát. Trong các hệ thống thuần nhất, người sử dụng chỉ cần có khả năng truy nhập vào tài nguyên của hệ thống; Khi phê chuẩn các truy nhập vào thư mục, file và mục dữ liệu, các hệ thống đa người dùng cần mức chi tiết kiểm soát cao hơn. Thứ ba, khi nói đến mức độ chi tiết của kiểm soát, người ta muốn nói đến mức độ điều khiển. Trong một số hệ thống, việc kiểm soát tất cả các file hệ thống xảy ra trong một vùng duy nhất (an toàn được đơn giản hoá), nhưng nếu có các lỗi xảy ra thì chúng chỉ tập trung trong vùng duy nhất này. Trong các hệ thống khác, an toàn phức tạp hơn nhiều, nhưng các

kiểm soát và các trách nhiệm được dàn trải trên các vùng khác nhau của hệ thống.

- Các thuộc tính được sử dụng cho kiểm soát truy nhập (*attributes used for access control*): Các quyết định an toàn dựa trên các thuộc tính của chủ thể/đối tượng (còn được gọi là các tân từ), dựa trên ngữ cảnh yêu cầu truy nhập. Các thuộc tính cơ bản là: vị trí, phân lớp chủ thể/đối tượng, thời gian, trạng thái một (hoặc nhiều) biến của hệ thống, lược sử truy nhập. Các chính sách an toàn phải được lựa chọn, tuỳ thuộc vào các nhu cầu kiểm soát.

- Tính toàn vẹn (*integrity*): Áp dụng các chính sách và mô hình an toàn xác định vào trong các môi trường, trong đó tính toàn vẹn là một mối quan tâm chính, ví dụ trong các môi trường cơ sở dữ liệu.

- Các quyền ưu tiên (*priorities*): Mâu thuẫn xảy ra giữa các nguyên tắc (về các chính sách an toàn) có thể tăng; Quyền ưu tiên có thể giải quyết tình trạng này. Một ví dụ điển hình trong các thành viên của nhóm, các quyền có thể mang tính cá nhân, hoặc có thể liên quan đến các thành viên của nhóm. Các quyền cá nhân có thể có độ ưu tiên cao hơn các quyền nhóm; hoặc quyền từ chối truy nhập có thể có độ ưu tiên cao hơn các quyền khác.

- Các đặc quyền (*privileges*): Các đặc quyền truy nhập chỉ ra: một chủ thể có thể truy nhập vào một đối tượng thông qua các chế độ (đọc, ghi, xoá, chèn). Các đặc quyền ngầm định phải được định nghĩa và việc sửa đổi các đặc quyền như thế nào cũng phải được định nghĩa. Các đặc quyền ngầm định có số lượng xác định và chúng phụ thuộc vào các chính sách được lựa chọn. Ví dụ, không tồn tại đặc quyền ngầm định cho các chính sách bắt buộc. Trong nhiều hệ thống, các phép đọc/ghi là các đặc quyền ngầm định, chúng được các chính sách tuỳ ý sử dụng kết hợp với các chính sách bắt buộc. Để quản lý đặc quyền, các phép trao/thu hồi đặc quyền phải được định nghĩa rõ ràng.

- Quyền (*Authority*): Một chính sách phải định nghĩa các kiểu vai trò, quyền và trách nhiệm khác nhau trong cùng một hệ thống. Các vai trò phổ biến là người sử dụng, người sở hữu, người quản trị an toàn. Ngoài ra còn có thêm các nhóm người sử dụng, điều này thực sự hữu ích khi những người sử dụng chia sẻ các yêu cầu truy nhập thông thường trong tổ chức (ví dụ, chia sẻ các nhiệm vụ thiết kế trong cùng một nhóm phát triển). Một chính sách phải xác

định tiêu chuẩn tổng hợp của nhóm, sự phân chia nhóm phù hợp với môi trường, thành viên cá nhân trong một hoặc nhiều nhóm và làm thế nào để giải quyết các mâu thuẫn có thể xảy ra với các đặc quyền do từng cá nhân nắm giữ. Các mức kiểm soát truy nhập khác nhau bao gồm: Mức 1- chỉ ra kiểm soát truy nhập hợp lệ mà các chủ thể sử dụng trên các đối tượng; Mức 2- chỉ ra kiểm soát truy nhập trên thông tin (thông tin này được sử dụng cho kiểm soát truy nhập); Mức 3- chỉ ra thông tin (thông tin này định nghĩa ai là người có thể truy nhập vào thông tin được sử dụng cho kiểm soát truy nhập). Trong các chính sách bắt buộc, mức 2 và 3 thường được gán cho người quản trị an toàn. Trong các chính sách tuỳ ý, mức 2 thường được gán cho người sở hữu tài nguyên. Nói chung, các vai trò phải được định nghĩa cùng với các quyền truy nhập, với các mức kiểm soát truy nhập khác nhau.

- Tính kế thừa (*inheritance*): Điều này chỉ ra việc sao chép các quyền truy nhập. Việc truyền lại các quyền truy nhập không xảy ra trong các chính sách tuỳ ý, nhưng lại xảy ra trong các chính sách bắt buộc.

Việc sử dụng các tiêu chuẩn, các nguyên tắc mức cao có thể được biên dịch thành dạng nguyên tắc, thích hợp cho giai đoạn hình thức hoá và giai đoạn thiết kế sau này. Việc lựa chọn các yêu cầu bảo vệ và các chính sách an toàn làm cơ sở cho giai đoạn thiết kế khái niệm tiếp theo.

2.4.3 Thiết kế khái niệm

Trong giai đoạn này, các yêu cầu và các chính sách an toàn (được định nghĩa trong giai đoạn trước đó) được hình thức hoá khái niệm. "Khái niệm" ở đây có nghĩa là chúng ta chưa quan tâm đến các chi tiết thực hiện. Đúng hơn là tập trung vào các khía cạnh ngữ nghĩa, tách bạch giữa các chính sách và các cơ chế. Mô hình an toàn khái niệm là một công cụ và nó được sử dụng cho việc hình thức hoá các yêu cầu và các chính sách.

Một mô hình an toàn khái niệm được định nghĩa thông qua:

- Nhận dạng các chủ thể và các đối tượng liên quan đến một quan điểm an toàn; Các chủ thể/đối tượng có chung một vai trò trong tổ chức được nhóm lại với nhau;

- Nhận dạng các chế độ truy nhập được trao cho các chủ thể khác nhau trên các đối tượng khác nhau; Nhận ra các ràng buộc có thể trên truy nhập;

- Phân tích việc thừa kế các quyền trên hệ thống, thông qua các đặc quyền trao/thu hồi. Phân tích này đặc biệt liên quan đến việc kiểm tra xem nguyên tắc đặc quyền tối thiểu có được tôn trọng hay không.

Từ các bước như trên, các yêu cầu được biểu diễn thành các bộ bốn, như sau: {*subject, access right, objects, predicate*}, trong đó tân từ (*predicate*) miêu tả các điều kiện truy nhập có thể. Các bộ bốn này biểu diễn các nguyên tắc truy nhập.

Các đặc trưng cơ bản của một mô hình an toàn khái niệm nên bao gồm:

- Biểu diễn các ngữ nghĩa của an toàn cơ sở dữ liệu. Điều này có nghĩa là tính bí mật và tính toàn vẹn của một mục dữ liệu được thể hiện trong các phạm vi của thông tin có trong mục này, phương thức sử dụng nó và mối quan hệ với các mục khác, bắt nguồn từ những thông tin thực tế mà mục dữ liệu này biểu diễn.

- Hỗ trợ việc phân tích các luồng quyền, có nghĩa là phân tích các kết quả khi trao/thu hồi quyền.

- Hỗ trợ người quản trị cơ sở dữ liệu, cho phép anh ta đưa ra các câu truy vấn trên tình trạng quyền hiện thời và kiểm tra các kết quả xảy ra do các thay đổi trên quyền. Tại mức khái niệm, các kiểm tra này dễ thực hiện hơn, so với tại mức cơ chế an toàn.

Mô hình ánh xạ các yêu cầu phi hình thức vào các nguyên tắc an toàn. Mô hình an toàn khái niệm cho phép biểu diễn tường minh các yêu cầu và các chính sách, đồng thời cho phép kiểm tra một số đặc tính của hệ thống an toàn.

Theo bộ tiêu chuẩn DoD, các mức phân loại hệ thống an toàn cao yêu cầu định nghĩa một mô hình cho hệ thống an toàn; Mô hình nên có một cấu trúc duy nhất, cấu trúc này tập hợp được tất cả các đặc tính của hệ thống và đảm bảo hỗ trợ hiệu quả cho các giai đoạn thiết kế, phê chuẩn và tổng hợp tài liệu sau này.

Mô hình an toàn khái niệm biểu diễn các chủ thể, các đối tượng, các phép toán và các chế độ truy nhập được trao, tuỳ thuộc vào các chính sách đã được định nghĩa. Mô hình của một hệ thống xác định phải như sau:

- Đầy đủ (*complete*): Mô hình đáp ứng được tất cả các yêu cầu an toàn đã được xác định ban đầu.

- Tương thích (*consistent*): Trong một hệ thống vẫn xảy ra sự không nhất quán, chẳng hạn như một người sử dụng trái phép không thể truy nhập trực tiếp vào một đối tượng, nhưng có thể đến được đối tượng thông qua đường dẫn khác, hoặc thông qua một loạt các phép toán. Trong trường hợp này, không gì có thể đảm bảo rằng các hoạt động được thực hiện trong hệ thống là các hoạt động được phép.

Mô hình phải đảm bảo rằng tất cả các yêu cầu được xác định, không mâu thuẫn với nhau và tồn tại tình trạng dư thừa. Các mâu thuẫn và/hoặc sự không rõ ràng trong các yêu cầu có thể được giải quyết, bằng cách tham khảo các nguyên tắc đã được biểu diễn trong các chính sách an toàn.

Hiện có rất nhiều mô hình an toàn, nhưng chúng vẫn chưa hoàn toàn tuân theo các yêu cầu dành cho một mô hình an toàn. Nguyên nhân là do việc lựa chọn các mô hình thường liên quan đến các vấn đề an toàn. Các mô hình có thể trừu tượng hoặc cụ thể, tuỳ thuộc vào việc sử dụng chúng cho các mục đích như: để chứng minh các đặc tính của hệ thống, hay chỉ là một hướng dẫn phát triển dành cho các hệ thống an toàn. Người ta cũng có thể kết hợp các mô hình thích hợp nhất, nhằm đưa ra một mô hình cơ bản đáp ứng được các yêu cầu ban đầu.Tuy nhiên, quá trình kết hợp các mô hình an toàn có thể gây ra tình trạng mất một số đặc tính an toàn của các mô hình tham gia. Chính vì vậy, chúng ta cần phê chuẩn và kiểm tra trước khi đưa ra mô hình cuối cùng.

Nói chung, việc chọn lựa các mô hình tuỳ thuộc vào kiểu của hệ thống, có nghĩa là kiểu của tài nguyên được bảo vệ; Nó cũng phụ thuộc vào các kiểm tra hình thức (các kiểm tra này đã được dự tính trước cho hệ thống).

2.4.4 Thiết kế lôgíc

Trong giai đoạn này, người ta sử dụng mô hình an toàn khái niệm, chuyển đổi nó thành một mô hình lôgíc. Mô hình này được DBMS xác định hỗ trợ. Ví dụ trong

một DBMS quan hệ, các kỹ thuật an toàn dựa vào câu truy vấn và khung nhìn thường xuyên được sử dụng cho kiểm soát truy nhập, đồng thời các nguyên tắc của mô hình khái niệm phải được chuyển đổi phù hợp, nhằm hỗ trợ các kỹ thuật này.

Hơn nữa, trong giai đoạn này, các nguyên tắc an toàn phải được xác định trên mô hình lôgíc, quan tâm đến các cơ chế mức OS và các chức năng mà các gói an toàn có thể đưa ra. Mục đích chính là để xác định các yêu cầu, chính sách và ràng buộc an toàn nào có thể được đáp ứng (tất cả chúng đã được biểu diễn trong mô hình an toàn khái niệm), thông qua các cơ chế an toàn hiện có, đồng thời xác định cơ chế nào cần được thiết kế phi hình thức.

Để đáp ứng mục đích trên, các nhà phát triển sử dụng thông tin (về các đặc tính an toàn đã được xác định trong giai đoạn khái niệm) để thiết lập các yêu cầu an toàn, chúng được cung cấp thông qua các cơ chế an toàn hiện có tại mức OS, hoặc mức DBMS, hoặc thông qua các gói an toàn (nếu có sẵn).

Trong khi sử dụng các cơ chế hiện có, có thể không tuân theo một số yêu cầu an toàn đã được xác định trong mô hình khái niệm, chính vì vậy, các nhà phát triển nên thiết kế các cơ chế đặc trưng, nhằm đáp ứng các yêu cầu.

2.4.5 Thiết kế vật lý

Trong giai đoạn này người ta quan tâm đến các chi tiết (liên quan đến việc tổ chức lưu giữ và thực hiện/tích hợp các mô hình). Bắt đầu từ mô hình an toàn lôgíc, tiến hành thiết kế chi tiết các cơ chế an toàn, cụ thể là thiết kế cấu trúc vật lý của các nguyên tắc truy nhập, quan hệ của chúng với các cấu trúc vật lý của cơ sở dữ liệu, các chế độ truy nhập liên quan và kiến trúc chi tiết của các cơ chế, tuân theo các yêu cầu và chính sách an toàn. Mục đích là gán mức bắt buộc chính xác cho từng nhiệm vụ an toàn.

2.4.6 Việc thực hiện của các cơ chế an toàn

Các nhà phát triển có thể sử dụng hữu ích một bộ các nguyên tắc. Nó trợ giúp cho họ trong việc chọn lựa và/hoặc thực hiện từ các cơ chế an toàn trộn ghép phi hình thức (*scratch ad hoc security mechanisms*), nhằm tránh các vấn đề nghiêm trọng sau này. Mục này trình bày các nguyên tắc như vậy và thảo luận về các vấn đề thực hiện.

Các nguyên tắc (dành cho việc chọn lựa/ thực hiện cơ chế) được liệt kê như sau:

- Tính kinh tế của các cơ chế (*economy of mechanisms*): Các cơ chế nên đơn giản (ở mức có thể được). Chính vì vậy, cần đơn giản hoá tính đúng đắn của việc thực hiện, cần kiểm tra chương trình trong trường hợp có lỗi xảy ra. Nói chung là mang lại các thuận lợi như : giảm bớt các chi phí, độ tin cậy cao hơn, việc kiểm tra và kiểm toán các giai đoạn của hệ thống dễ dàng hơn.

- Hiệu quả (*efficiency*): Các cơ chế nên hiệu quả, một phần là do chúng thường được gọi trong thời gian chạy. Một hướng tiếp cận dựa vào nhân không thoả mãn hoàn toàn yêu cầu này do quá tải hoặc các gánh nặng về hiệu năng.

- Độ tuyến tính của các chi phí (*linearity of costs*): Sau giai đoạn cài đặt (thiết lập), các chi hoạt động nên cân xứng với việc sử dụng thực tế của cơ chế.

- Tách bạch đặc quyền và các trách nhiệm (*privilege separation and responsibility*): Bất cứ khi nào có thể, chúng ta nên phân tầng một số cơ chế kiểm soát và việc thực hiện truy nhập phụ thuộc vào nhiều điều kiện (độ phức tạp sẽ là một rào cản an toàn tốt hơn). Chúng ta có thể đảm bảo được việc tách bạch đặc quyền, thông qua việc sử dụng các cơ chế khác nhau và sử dụng lặp đi lặp lại cùng một cơ chế, như trong các hệ thống phân tán (đây là nơi có một số mức mật khẩu).

- Đặc quyền tối thiểu (*minimum privilege*): Nên giới hạn mức đặc quyền tối thiểu đối với các chương trình và người sử dụng. Chúng ta cần quan tâm đến nguyên tắc "cần-để-biết" khi lựa chọn chính sách và áp dụng nguyên tắc này cho các thành phần của hệ thống. Các thuận lợi như sau:

  - Hạn chế lỗi (*error confinement*): Cần tối giản các hậu quả do các thành phần sai sót gây ra; Trong thực tế, hạn chế phần bộ nhớ mà một thành phần có thể truy nhập vào, nhằm hạn chế các thiệt hại có thể xảy ra;

  - Duy trì (*maintenance*): Chúng ta có thể dễ dàng ước tính các hậu quả của việc sửa đổi thành phần, trên một số lượng giới hạn các thành phần.

  - Ngăn chặn con ngựa thành Tơroa (*defence from Trojan horses*): Hạn chế các hoạt động của con ngựa thành Tơroa;

- Dàn xếp toàn bộ (*complete mediation*): Mỗi truy nhập vào một đối tượng phải tuân theo kiểm soát quyền.

- Thiết kế phổ biến (*known design*): Nên phổ biến rộng rãi các kỹ thuật an toàn đã được phê chuẩn. Thành phần bí mật phải dựa vào khoá và mật khẩu (liên quan đến các kỹ thuật xử lý khoá như: sinh, lưu giữ và phân phối khoá).

- An toàn thông qua ngầm định (*security by default*): Nếu người sử dụng không quyết định các tuỳ chọn thì các tuỳ chọn bảo vệ thường được đặt ngầm định. Các quyết định truy nhập nên dựa vào các quyền trao (*grant*), hơn là các quyền từ chối (*denial*). Thông thường, một chính sách hệ thống khép kín thường được ưa chuộng hơn, vì nó an toàn hơn hơn một chính sách mở.

- Các cơ chế phổ biến tối thiểu (*minimum commom mechanisms*): Theo nguyên tắc này, việc thiết kế phải khuyến khích tính độc lập lẫn nhau giữa các cơ chế, nghĩa là hoạt động đúng đắn của cơ chế này không nên phụ thuộc vào hoạt động đúng đắn của cơ chế khác.Ví dụ, nên sử dụng các cơ chế khác nhau cho các kiểm soát vật lý và lôgíc, có nghĩa là hạn chế một cách tối đa các hậu quả của việc truy nhập trái phép (do kẻ xâm nhập gây ra).

- Khả năng chấp nhận mang tính tâm lý (*psycholôgical acceptability*): Việc sử dụng các cơ chế phải dễ dàng, cho phép người dùng sử dụng chúng một cách phù hợp. Nên tránh các hạn chế không cần thiết. Việc xác định các quyền truy nhập cũng phải dễ dàng, nhờ đó mới khuyến khích người sử dụng tham gia bảo vệ.

- Tính mềm dẻo (*flexibility*): Một cơ chế an toàn nên tuân theo các chính sách khác nhau. Tính mềm dẻo của cơ chế cũng bao hàm cả việc duy trì bảo vệ chống lại các tấn công chủ ý hoặc vô ý. Do vậy, thiết kế cần đảm bảo được các hoạt động đúng đắn và liên tục của cơ chế, ngay cả khi các biến cố xảy ra do các tấn công chủ ý hoặc vô ý. Đồng thời, trong các điều kiện xấu nhất, cơ chế vẫn làm việc.

- Sự cách ly (*isolation*): Cơ chế an toàn phải chứng minh được rằng nó phù hợp với các yêu cầu chính sách an toàn. Phương pháp luận phát triển hình thức là một công cụ hiệu quả cho mục đích này.

- Tính đầy đủ và tương thích (*completeness and consistancy*): Cơ chế an toàn phải đầy đủ (có nghĩa là nó tuân theo toàn bộ các đặc tả thiết kế) và nhất quán (có nghĩa là không tồn tại các mâu thuẫn vốn có). Việc bảo vệ phải được xác định, thông qua một tập hợp các đặc tả "khẳng định" và một tập hợp các đặc tả "phủ định".

- Khả năng quan sát (*observability*): Cần có khả năng kiểm soát cơ chế và các tấn công chống lại cơ chế. Điều này cho phép chỉ ra các điểm yếu hoặc các lỗi thiết kế. Các file nhật ký, vết kiểm toán, sổ nhật ký và các công cụ tương tự được sử dụng để lưu lại các hoạt động của cơ chế.

- Vấn đề do bỏ sót (*problem of residuals**): Phần bị bỏ sót là các đoạn thông tin. Một tiến trình đã sử dụng chúng và có thể lưu giữ chúng trong bộ nhớ mặc dù tiến trình đã kết thúc. Dữ liệu có thể bị lộ nếu các chủ thể trái phép có thể kiểm tra các phần bị bỏ sót. Cơ chế an toàn thích hợp thường loại bỏ các phần bị bỏ sót.

- Khả năng không nhìn thấy được của dữ liệu (*invisibility of data*): Người sử dụng trái phép không được biết các thông tin về cấu trúc (lược đồ), hoặc sự tồn tại của đối tượng cùng với các nội dung có trong đối tượng, để tránh suy diễn (ví dụ, đọc tên của các đối tượng trong hệ thống).

- Hệ số làm việc (*work factor*): Việc phá vỡ một cơ chế an toàn phải là một công việc khó khăn, đòi hỏi nhiều nỗ lực. Chúng ta có thể đánh giá chất lượng của một cơ chế, thông qua việc so sánh các nỗ lực (cần thiết để vượt qua cơ chế) với tài nguyên mà một kẻ xâm nhập tiềm ẩn có thể có. Nói chung, mức bảo vệ càng tốt thì các nỗ lực cần thiết để phá vỡ càng cao. Các nỗ lực phá vỡ thường khó xác định. Tuy nhiên, vẫn có một số phương thức phá vỡ cơ chế an toàn (ví dụ, lỗi phần cứng hoặc lỗi thực thi).

- Các bẫy chủ ý (*intentional traps*): Chúng ta có thể thiết kế một cơ chế với các lỗi chủ ý bên trong, sau đó kiểm soát chúng trong thời gian thực của hệ thống, để phát hiện ra các cố gắng xâm nhập có thể, nhằm giám sát các hành vi của kẻ xâm nhập. Giải pháp này thực sự hữu ích, thông qua nó chúng ta có thể biết được thông tin (về các kiểu tấn công) và biết được hành vi của kẻ xâm nhập. Tuy nhiên, thủ tục này có thể vi phạm các luật và dự luật hiện hành. Chúng ta cần kiểm tra và thận trọng khi sử dụng thủ tục này.

- Xác định tình trạng khẩn cấp (*emergency measures*): Chúng ta nên thiết kế cơ chế cùng với các phương thức "mất khả năng làm việc" (Trong trường hợp xây dựng lại hoặc định lại cấu hình cho hệ thống, nhưng cần sử dụng đến các yêu cầu hoặc các sự cố mang tính tổ chức đặc thù). Nói chung, để hỗ trợ cho trường hợp này và làm tăng độ mềm dẻo của cơ chế, nên cho phép một số người sử dụng tin cậy tạm thời ngừng kiểm soát, hoặc sửa đổi tạm thời các phương thức kiểm soát.

- Phần cứng an toàn (*secure hardware*): Yêu cầu này dành cho hệ thống được bảo vệ. Phần cứng phải tin cậy, bởi vì kẻ xâm nhập có thể dễ dàng sử dụng các lỗi phần cứng/phần mềm nhằm vượt qua kiểm soát an toàn. Có thể thiếu các phương tiện lưu giữ nếu chỉ dựa vào cơ chế kiểm soát truy nhập.

- Ngôn ngữ lập trình (*programming language*): Chúng ta cần quan tâm đến ngôn ngữ lập trình. Chương trình biên dịch ngôn ngữ (dùng để lập trình các cơ chế) phải tin cậy. Các lập trình viên có kỹ năng góp phần đảm bảo tính tin cậy, cũng như khả năng duy trì và phát hiện lỗi của hệ thống. Trong khi lập trình cơ chế an toàn, chúng ta phải tuân thủ một cách nghiêm ngặt các đặc tả của mô hình. Trong thực tế, các điểm yếu dễ bị tấn công có thể có nguồn gốc từ các thay đổi thiết kế trong giai đoạn lập trình, do các bắt buộc về hiệu năng hoặc nhu cầu giảm chi phí phát triển. Các thay đổi cần thiết cần được nhận dạng rõ ràng và tối thiểu hóa. Thêm vào đó, trong trường hợp sửa đổi chương trình (nó không chỉ xảy ra trong giai đoạn ban đầu mà còn xảy ra trong quá trình duy trì hệ thống), các đặc tả nên được cập nhật song song.

- Tính đúng đắn (*correctness*): Các cơ chế phải làm sáng tỏ chính xác mô hình. Hoàn toàn không nên sử dụng các cơ chế không đúng đắn, bởi vì chúng có thể đưa ra các bảo vệ sai lầm. Các cơ chế không đúng đắn có thể gây ra tình trạng: thứ nhất, từ chối các hoạt động hợp pháp, thứ hai, trao các truy nhập trái phép. Với tình trạng thứ hai, hệ thống sẽ gặp nguy hiểm, do không đảm bảo được tính bí mật/toàn vẹn dữ liệu.

Chúng ta có thể tiến hành phát triển các cơ chế trộn ghép, thông qua phần cứng, phần mềm và phần sụn. Khi chọn lựa các kỹ thuật trong giai đoạn thực hiện, dàn xếp phần cứng/phần mềm một cách thích hợp, quan tâm đến độ phức tạp, kích cỡ và hiệu quả mà một hệ thống cuối cùng yêu cầu. Tốt hơn, chúng ta nên tiến hành chọn lựa trên tất cả các cơ chế phần cứng, chúng phải chứng minh được tính tin

cậy, an toàn và có thể chịu trách nhiệm toàn bộ về an toàn của hệ thống. Thông qua cách khác, các cơ chế phần mềm có thể hoạt động như là các trình thông dịch (*interpreter*) chỉ dẫn người dùng. Một chọn lựa tốt là thực hiện các cơ chế phần cứng ở một mức độ nào đó, còn lại là phần mềm.

Trong giai đoạn này cần các mô hình kiến trúc hệ thống an toàn. Các mô hình kiến trúc biểu diễn mối quan hệ giữa các modul hệ thống và các giao diện của module, có nghĩa là, các module truyền thông như thế nào khi các chức năng an toàn được thực hiện. Các mô hình này cũng minh hoạ các chức năng an toàn được gán cho DBMS, cho OS như thế nào và nhấn mạnh vai trò của TBC trong các cơ chế của hệ thống. Mô hình kiến trúc an toàn DBMS đã được trình bày ở trên. Theo kiến trúc này, các phương pháp khác nhau có thể được chấp nhận, tuỳ thuộc vào các yêu cầu xác định của môi trường.

Để thực hiện các cơ chế phần mềm, cần nhớ rằng việc chuyển đổi mô hình an toàn hình thức sang cơ chế thực thi tương ứng là gián tiếp. Nói chung trong thực tế, mô hình không cung cấp trực tiếp các đặc tả thực thi. Vì vậy, chúng ta nên tiến hành ánh xạ giữa mô hình hình thức và mức thực thi, thông qua các chuyển đổi thích hợp, từ các đặc tả hình thức tới các chương trình cụ thể của cơ chế.

Trong giai đoạn này, các vấn đề liên quan đến hiệu năng của hệ thống cũng cần được quan tâm. Các kiểm soát an toàn yêu cầu tăng thêm thời gian trong quá trình xử lý dữ liệu: các tham số về số lượng, chẳng hạn như thời gian đáp ứng, thời gian/không gian lưu trữ hoặc cập nhật dữ liệu, cũng như các tham số về chất lượng (ví dụ, tính mềm dẻo, tính tương thích, tính hoán đổi sang các môi trường mới, khả năng khôi phục) cần được quan tâm.

2.4.7 Việc kiểm tra và thử nghiệm

Đúng hơn là dành cho việc phát triển hệ thống phần mềm, một biểu diễn của hệ thống trong giai đoạn đầu tiên của phương pháp phát triển phải được chuyển đổi một cách chính xác sang biểu diễn trong giai đoạn chi tiết tiếp theo và cuối cùng là một sản phẩm phần mềm đúng đắn. "Tính đúng đắn" của phần mềm an toàn phát triển bao hàm nhiều thứ khác nhau, tuỳ thuộc vào mức tin cậy mong muốn đối với phần mềm. Nói chung, mục đích của giai đoạn này là kiểm tra các yêu cầu và các chính sách an toàn. Việc kiểm tra này được thực hiện thông qua sản phẩm phần

mềm. Để làm được điều này cần có sẵn các phương pháp hình thức và phi hình thức, dựa vào hoặc không dựa vào các ký hiệu toán học.

Các phương pháp phi hình thức dựa trên :

1) Kiểm soát chéo các yêu cầu/chương trình nguồn, hoặc các yêu cầu/các hành vi tại thời gian chạy (để chứng minh rằng phần mềm đã tuân theo mọi yêu cầu an toàn được định nghĩa trong giai đoạn phân tích);

2) Duyệt lại chương trình phần mềm để phát hiện ra các lỗi/các mâu thuẫn (tính không nhất quán);

3) Phân tích hành vi của chương trình, tuỳ thuộc vào các tham số khác nhau, nhằm kiểm tra các đường dẫn thực hiện khác nhau và các biến thể tương ứng của các tham số;

4) Thông qua thử nghiệm, gỡ rối;

Nói chung, các phương pháp phi hình thức có thể được áp dụng một cách nhanh chóng, không cần định nghĩa trước mô hình an toàn hình thức. Các phương pháp phi hình thức có thể xác định hành vi của phần mềm trong các trường hợp cụ thể, nhưng nó không thể chỉ ra cấm thực hiện các hoạt động trái phép trong hệ thống.

Các phương pháp hình thức tinh xảo hơn, chúng dựa vào các ký hiệu và các phương pháp toán học. Cần phân tích mô hình hình thức của các yêu cầu an toàn nhằm đảm bảo tính đúng đắn, thông qua các cuộc thử nghiệm đúng đắn. Các đặc tả hình thức mức cao tinh xảo hơn các đặc tả mức trung gian và các đặt tả mức thấp. Các kỹ thuật này có thể chứng minh mô hình là cơ chế an toàn, thông qua việc chứng minh tính đúng đắn của các đặc tả hình thức. Cần phải chứng minh: mỗi đặc tả mức trung gian nhất quán với đặc tả mức trước đó. Việc chứng minh cứ tiếp diễn cho đến khi kiểm tra được các đặc tả mức cao nhất quán với mô hình an toàn hình thức. Các ngôn ngữ đặc tả và các công cụ kiểm tra đang được phát triển cho các hệ thống an toàn thiết yếu.

GIẢI PHÁP BẢO VỆ DỮ LIỆU CSDL

Nội dung nghiên cứu phần này nhằm mục đích xây dựng giải pháp bảo vệ dữ liệu CSDL khi lưu chuyển trên kênh giữa DataBase Server và DataBase Client dựa trên mô hình Winsock. Mô hình mạng Winsock là một mô hình được sử dụng rộng rãi ngày nay. Do vậy định hướng nghiên cứu vào mô hình này rất có ý nghĩa thực tiễn.

Trong phần tài liệu này sẽ trình bầy một số vấn đề sau:

- Mô hình Windows Socket,
- Mô hình SecureSocket,
- Thiết kế chương trình thử nghiệm.

MÔ HÌNH WINSOCK

1. Winsock Model

Để thực hiện mục tiêu bảo vệ thông tin trong CSDL, chúng tôi lựa chọn mô hình mạng Winsock để tiếp cận đến mục tiêu. Sở dĩ chúng tôi lựa chọn mô hình Winsock vì những lý do sau:

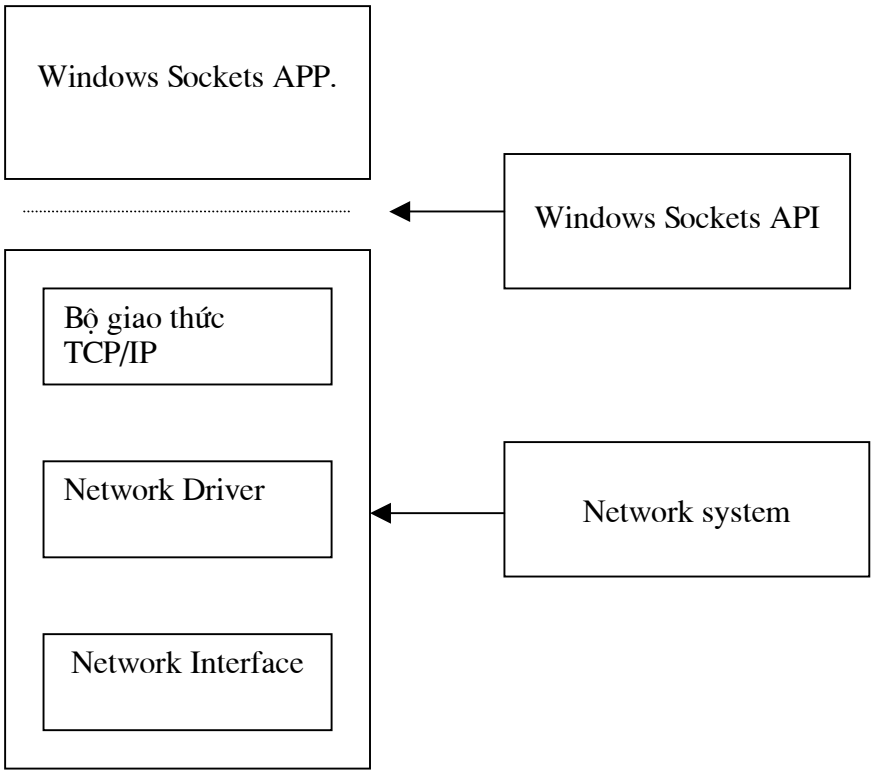
- Winsock là một mô hình được sử dụng rộng rãi hiện nay.
- Winsock là một mô hình mở, cho phép ta can thiệp để đạt được những mục tiêu mong muốn.

Một mô hình mạng mà chúng ta đã biết được xem như một kiến trúc mạng chuẩn là mô hình mạng OSI. Mục đích của mô hình này là đồng nhất và định nghĩa một tập các hàm chung để xử lý mọi truyền thông mạng giữa các máy tính nối mạng với nhau.

Mô hình mạng Winsock cũng được xây dựng trên tinh thần của mô hình mở OSI tuy nhiên có những điểm khác biệt. Mô hình mạng Winsock được tổ chức thành các phần sau:

- Winsock application: Cung cấp những chức năng của các tầng 5,6,7 trong mô hình OSI.
- Network system: cung cấp các chức năng của các tầng 1,2,3,4 trong mô hình OSI.
- Winsock API: cho phép tầng trên truy nhập các dịch vụ của tầng dưới.

Ta có thể minh hoạ mô hình mạng Winsock trong hình sau.



Mô hình mạng Winsock

Winsock APP. là một chương trình ứng dụng cùng với giao diện người dùng. Nó cũng có thể là một thư viện động DLL trung gian cùng với API mức cao hơn và các ứng dụng của nó. Trong mô hình Winsock ta xem một ứng dụng bất kỳ mà truy nhập Winsock DLL như là một ứng dụng của Winsock.

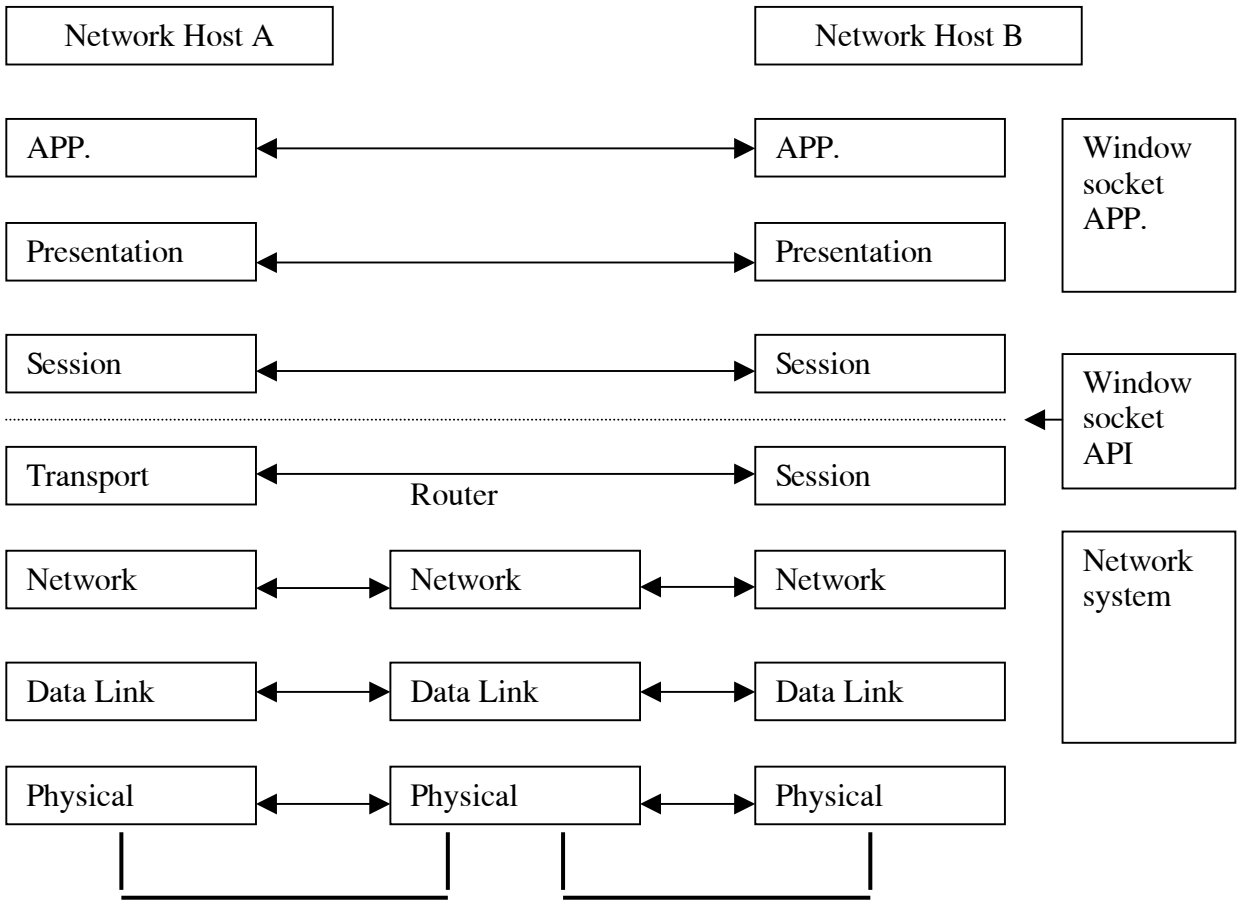
Winsock API (WSA) cung cấp truy nhập tới Network system và các ứng dụng của Winsock sử dụng các dịch vụ của hệ thống để gửi và nhận thông tin.

Network system truyền và nhận dữ liệu mà không hề quan tâm đến nội dung và ngữ nghĩa của nó. Khi Winsock APP. gửi một khối dữ liệu, Network system có thể chia khối dữ liệu đó thành nhiều đoạn khác nhau và hợp nhất lại tại đầu nhận trước khi chuyển giao. Nó cũng có thể xem dữ liệu như một dòng các bytes và yêu cầu ứng dụng hợp nhất lại sau khi chuyển giao. Dữ liệu được xem như thế nào phụ thuộc vào các dịch vụ tầng vận tải được yêu cầu. Nhưng trong bất kỳ trường hợp nào thì Network system cũng chuyển giao dữ liệu mà không quan tâm đến nội dung và ngữ nghĩa của dữ liệu.

Mô hình mạng Winsock về bản chất là dạng đơn giản của mô hình OSI. Tuy vậy, các tầng chức năng của mô hình OSI vẫn tồn tại trong mô hình Winsock ở mức quan niệm.

Windows Socket độc lập với giao thức cho nên nó có thể thích nghi với nhiều bộ giao thức khác nhau. Nó cũng độc lập với thiết bị mạng cho nên các ứng dụng trên Windows Socket có thể chạy trên bất kỳ thiết bị mạng nào mà Windows system hỗ trợ. Windows socket có thể hỗ trợ một số bộ giao thức khác nhau đồng thời.

Ứng dụng của Windows socket liên hệ với ứng dụng chạy trên một máy tính khác có thể minh hoạ trong hình sau.

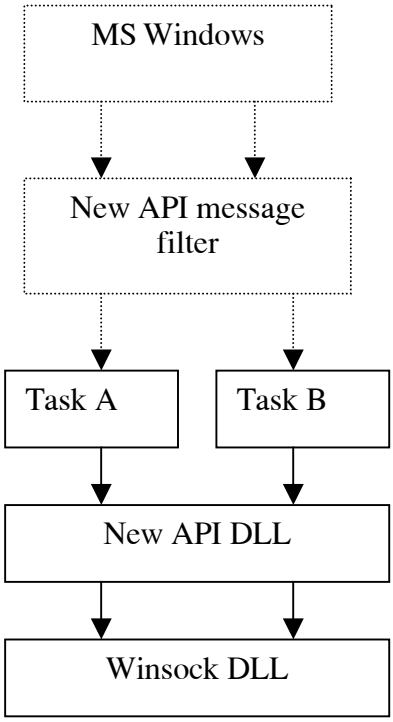


Truyền thông giữa các tầng đồng mức

Các tầng đồng mức hội thoại với nhau sử dụng cùng giao thức đó là tập các qui tắc để giao tiếp giữa các tầng đồng mức. Các qui tắc mô tả những yêu cầu và phúc đáp phù hợp với trạng thái hiện tại. Hội thoại giữa hai tầng đồng mức độc lập với hội thoại giữa các tầng đồng mức khác. Hội thoại giữa hai tầng đồng mức chỉ là quan niệm chứ không phải dòng dữ liệu thực tế.

2. Xây dựng các DLL trên Winsock

Toàn bộ dòng thông tin trên mạng trong các Platform Windows đều chuyển qua Winsock. Vấn đề đặt ra là làm thế nào để có thể khống chế được dòng thông tin này để phục vụ cho các mục tiêu riêng biệt. Can thiệp trực tiếp vào các Modul trong Winsock là một việc làm khó có thể thực hiện được bởi đối với những người phát triển ứng dụng thì Winsock chỉ như một chiếc hộp đen. Chúng ta chỉ có thể biết được giao diện với Winsock mà thôi. Vậy cách tiếp cận là như thế nào. Chúng tôi tiếp cận theo kiểu xây dựng một API mới trên Windows Socket API. Dòng thông tin trước khi chuyển qua Winsock sẽ qua một tầng mới do ta xây dựng và ở tầng này chúng ta có thể khống chế được dòng thông tin mạng.



Dòng thông tin với API DLL mới

Khi xây dựng một tầng mới trên tầng Winsock có nhiều kỹ thuật phải giải quyết. Một trong những kỹ thuật cần phải quan tâm đó là xử lý các message được gửi từ Winsock cho ứng dụng. Nếu không chặn được dòng message này thì không thể điều khiển được quá trình truyền thông giữa ứng dụng tại client và phần ứng dụng tại server. Chẳng hạn khi ta chèn thêm một packet vào dòng packet của ứng dụng. Nếu ta không xử lý được các message gửi từ Winsock cho ứng dụng thì hầu như chắc chắn connection giữa client và server sẽ bị huỷ bỏ và quá trình trao đổi thông tin

giữa client và server sẽ bị huỷ giữa chừng. Kỹ thuật được chọn xử lý ở đây là sử dụng kỹ thuật subclass. Mục tiêu chính của nó là chặn toàn bộ các message gửi từ Winsock cho ứng dụng, xử lý những message cần thiết và trả lại những message của ứng dụng cho ứng dụng xử lý.

3. Sự liên kết giữa Client và Server trong mô hình Winsock

Để các socket tại Client và Server có thể giao tiếp được với nhau thì chúng phải có cùng kiểu. Các ứng dụng Client phải có khả năng xác định và nhận ra socket tại server. Ứng dụng tại server đặt tên socket của nó và thiết lập những đặc tính để nhận diện của nó. Do vậy mà client có thể tham chiếu nó. Mỗi tên socket cho TCP/IP bao gồm địa chỉ IP, số hiệu cổng cũng như giao thức. Client có thể sử dụng các hàm dịch vụ của Windows Socket để tìm ra số hiệu cổng của server, địa chỉ IP của server nếu biết được tên của server. Khi client socket liên hệ thành công với server socket thì hai tên của chúng kết hợp lại để tạo thành một liên kết. Mỗi liên kết có 5 thành phần sau:

- Giao thức,
- Địa chỉ IP của Client,
- Số hiệu cổng của Client,
- Địa chỉ IP của Server,
- Số hiệu cổng của Server.

Khi một socket được mở, nó có những đặc tính chưa đầy đủ. Để hoàn tất đặc tính của nó, ứng dụng mạng phải gán cho nó một tên và liên kết nó với một socket khác. Các phép toán send và receive của socket rất giống với các phép toán read và write tới file. Khi close một socket có nghĩa là giải phóng nó khỏi ứng dụng và trả về cho hệ thống để có thể sử dụng cho việc khác.

Socket là điểm cuối của một liên kết truyền thông, nó được tạo ra bởi phần mềm và cho phép ứng dụng mạng đăng nhập vào mạng. Cả client và server đều đòi hỏi socket để truy nhập mạng. Mở một socket thông qua gọi hàm socket() có khai báo hàm như sau:

```
SOCKET PASCAL FAR socket(int af,                /*Bộ giao thức*/
                         int type,              /*kiểu giao thức*/
                         int protocol);         /*tên giao thức*/
```

ứng dụng Windows socket socket()

socket handle	
1. Protocol	
2. local IP address 3. local port	4. remote IP address 5.remote port

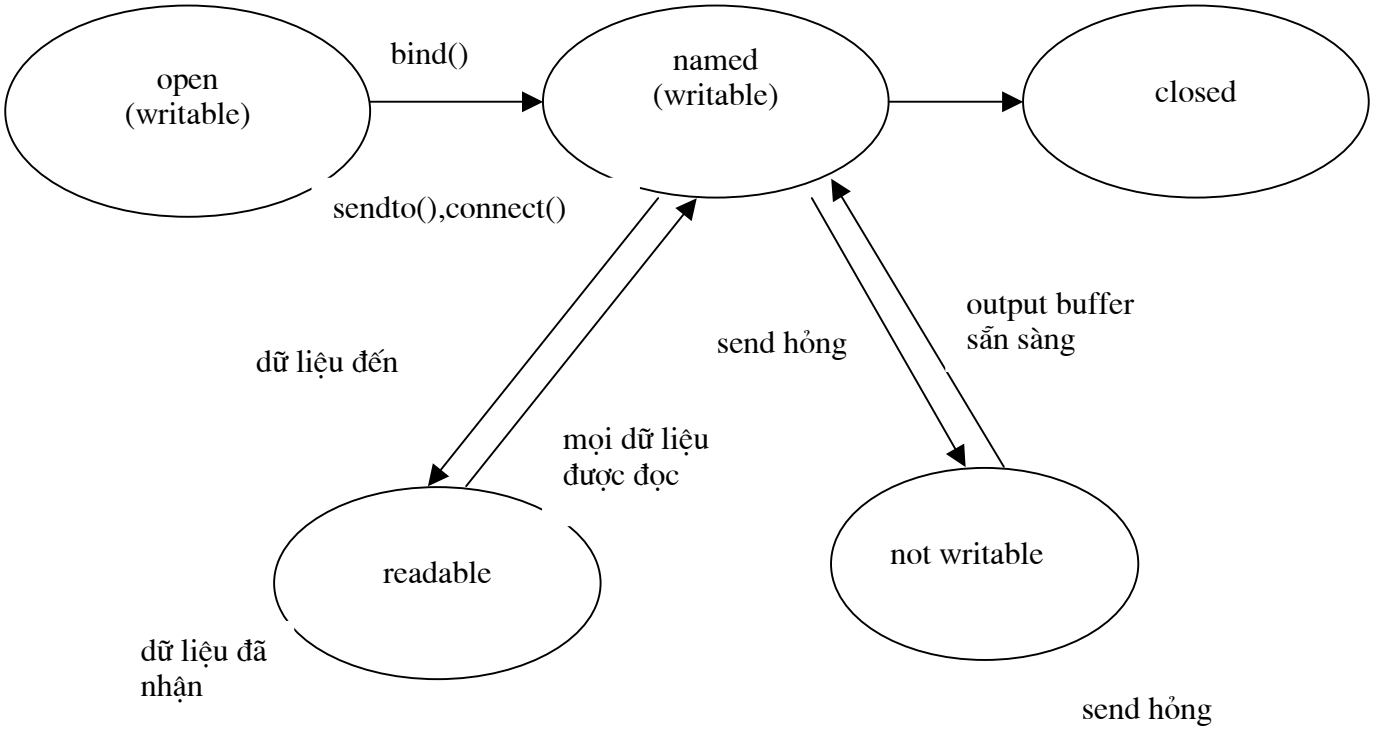
Server cần phải chuẩn bị socket của mình để nhận dữ liệu còn client cần chuẩn bị socket của mình để gửi dữ liệu. Khi việc chuẩn bị xong sẽ tạo ra một liên kết giữa các socket của client và server. Mỗi liên kết là duy nhất trên mạng. Khi liên kết giữa các socket được thiết lập có nghĩa client và server nhận diện được nhau và có thể trao đổi dữ liệu được với nhau.

4. Các trạng thái của socket

Trong phần này chúng tôi sẽ trình bầy các phương pháp khác nhau phát hiện trạng thái hiện thời cuả socket và các phép chuyển tới những trạng thái mới. Trạng thái hiện thời của socket xác định các phép toán mạng nào sẽ được tiếp tục, các phép toán nào sẽ bị treo lại và những phép toán mạng nào sẽ bị huỷ. Mỗi socket có một số hữu hạn các trạng thái có thể và winsock API định nghĩa các điều kiện cho phép chuyển giữa các sự kiện mạng và các lời gọi hàm của ứng dụng. Có hai kiểu socket: datagram socket và stream socket. Mỗi kiểu socket có những trạng thái và những phép chuyển khác nhau.

4.1. Các trạng thái của socket kiểu datagram

Sơ đồ trạng thái của socket kiểu datagram có thể biểu diễn trong hình sau.

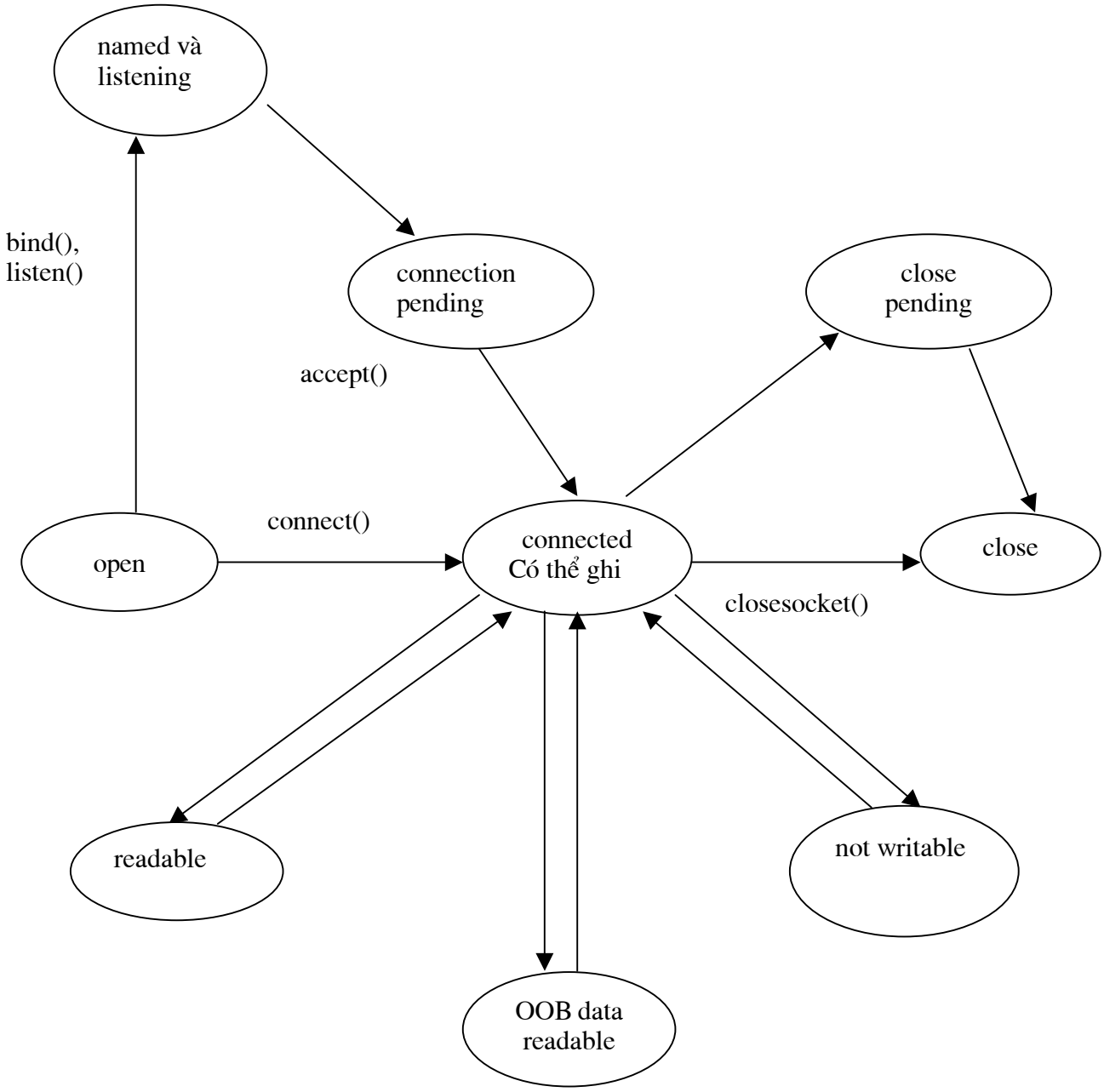


Sơ đồ trạng thái của socket kiểu datagram

Sơ đồ trên minh hoạ tất cả các trạng thái mà ta có thể xác định bằng chương trình. Nó cũng chỉ ra các phép chuyển xẩy ra khi ứng dụng thực hiện lời gọi hàm của winsock hoặc nhận các packet từ các máy ở xa. Trong sơ đồ này cũng chỉ ra rằng với socket kiểu datagram thì có thể ghi ngay được ngay sau khi nó được mở và nó có thể đọc ngay khi nó được định danh, ứng dụng có thể tiến hành gửi dữ liệu ngay sau lời gọi hàm socket()...

4.2. Các trạng thái của socket kiểu stream

Ta có thể minh hoạ các trạng thái của socket kiểu stream trong sơ đồ trạng thái sau.



Sơ đồ trạng thái của socket kiểu stream

ở trạng thái open socket được tạo ra thông qua lời gọi hàm socket() nhưng tại thời điểm này socket chưa được xác định có nghĩa nó chưa được liên kết với một địa chỉ mạng cục bộ và một số hiệu cổng.

ở trạng thái named và listening: lục này socket đãđược xác định và sẵn sàng đón nhận các yêu cầu kết nối.

connect pending: yêu cầu kết nối đã được nhận và chờ ứng dụng chấp nhận kết nối.

connected: liên kết được thiết lập giữa socket cục bộ và socket ở xa. Lúc này có thể gửi và nhận dữ liệu.

readable: Dữ liệu đã nhận được bởi mạng và sẵn sàng cho ứng dụng đọc (có thể đọc bằng các hàm recv() hoặc recvfrom())

XÂY DỰNG SOCKET AN TOÀN

Chúng tôi phát triển một giao diện tại tầng giao vận cho truyền thông TCP/IP được gọi là Secure Socket để phục vụ cho mục tiêu nén và mã hoá dữ liệu truyền qua Internet và các mạng PSTN.

Secure Socket được cài đặt tại các trạm, Server và FireWall để đảm bảo an toàn và truyền thông tốc độ cao giữa trạm và các máy chủ.

Secure Socket cung cấp giao diện lập trình ứng dụng Winsock chuẩn cho các ứng dụng TCP/IP chẳng hạn như Web Browser, telnet, ftp mà không cần bất kỳ sự thay đổi nào đối với các trình ứng dụng và TCP/IP.

Trong tài liệu này sẽ mô tả cấu trúc của Secure Socket, cách thức làm việc và lợi ích đối với môi trường truyền thông từ xa.

Trong các cơ quan có nhiều máy cá nhân, Server được kết nối với mạng LAN của cơ quan. Các nhân viên trong cơ quan có thể truy nhập CSDL tại Server từ các máy cá nhân trên bàn làm việc của mình hoặc từ các máy ở xa thông qua mạng Internet.

Có hai rủi ro chính khi truy nhập dữ liệu từ xa qua Internet:

- Dữ liệu có thể bị đánh cắp,
- Nghe trộm hoặc thay đổi.

Chúng tôi sẽ đề xuất một phương pháp truyền thông có nén và mã hoá dữ liệu môi trường tính toán từ xa. Sử dụng phương pháp này, chúng tôi phát triển chương trình mã hoá và nén dữ liệu được gọi là Secure Socket có thể cung cấp khả năng truy nhập từ xa hiệu quả và an toàn qua Internet và PSTN mà không cần thay đổi thiết bị mạng, phần mềm truyền thông hoặc phần mềm ứng dụng.

1. Các yêu cầu khi thiết kế

- ***Khả năng thích nghi***: Các đặc tính an toàn cần phải làm việc được với mọi platform phần cứng, phần mềm, các thủ tục truyền thông hoặc các thiết bị truyền thông khác nhau. Ví dụ IP an toàn mã hoá dữ liệu truyền giữa các router chỉ đảm bảo an toàn cho những dữ liệu truyền qua những router đã cài đặt IP an toàn. Mã hoá dữ liệu end-to-

end có thể giải quyết vấn đề này mà không cần phải chú ý đến những chức năng của router.

- ***Trong suốt***: Không cần phải có những thay đổi trong các trình ứng dụng bởi vì khả năng thay đổi những ứng dungj đang tồn tại hiện nay là hầu như không thể.

- ***Có khả năng mở rộng***: Có nhiều thuật toán mã hoá và nèn dữ liệu đang tồn tại và những thuật toán mới sẽ xuất hiện trong tương lai. Do vậy, khả năng lựa chọn thuật toán là cần thiết và các Modul xử lý chúng nên độc lập với các modul khác để chúng có thể thay thế được dễ dàng.

- ***Dể cài đặt***: Các modul an toàn có thể cài đặt trên những PC và Server một cách dễ dàng mà không cần thay đổi hệ điều hành.

- ***Hiệu quả:*** Khả năng thông qua của kênh không được giảm bởi những chi phí do nén và mã hoá dữ liệu. Việc nén dữ liệu có thể tăng ảo khả năng thông qua của kênh.

2. Kiến trúc

Secure Socket giải quyết được vấn đề cho phép người dùng từ xa có thể truy nhập mạng làm việc thông qua Internet hoặc mạng điện thoại công cộng một cách tin cậy.

Hình 1. Cho xem một truy nhập từ xa từ một PC ở xa mà ở đó Secure Socket đã được cài đặt. Có hai dạng truy nhập từ xa:

- Dạng thường được dùng trong các văn phòng nhỏ mà ở đó người dùng ở xa kết nối với Server ứng dụng bằng Secure socket được cài đặt qua Remote Acces Server. Toàn bộ dữ liệu được trao đổi giữa PC ở xa và Server sẽ được nén , mã hoá, xác thực .

- Dạng được dùng trong các mạng xí nghiệp. Trong các mạng này, người dùng kết nối tới Firewall đã cài đặt Secure socket. Toàn bộ dữ liệu được truyền giữa PC ở xa và Firewall được nén, mã hoá và xác thực. Firewall sau đó, giải mã, giải nén dữ liệu và trao đổi dữ liệu với Server ứng dụng.

Secure socket bao gồm thư viện liên kết động tầng giao vận. Nó được đặt giữa các chương trình ứng dụng và TCP/IP, các trình tiện dụng tương tác với người dùng. Tại các PC client thì Winsock

là giao diện lập trình ứng dụng chuẩn cho TCP/IP. Chúng ta có thể thực hiện nén, mã hoá và xác thực dữ liệu mà không cần thay đổi phần mềm ứng dụng hoặc TCP/IP. Hình 2 cho xem cấu trúc Secure socket chặn các lệnh của Winsock.

3. Thực hiện

3.1. Phương pháp chặn

Chặn các lệnh của Winsock như sau:

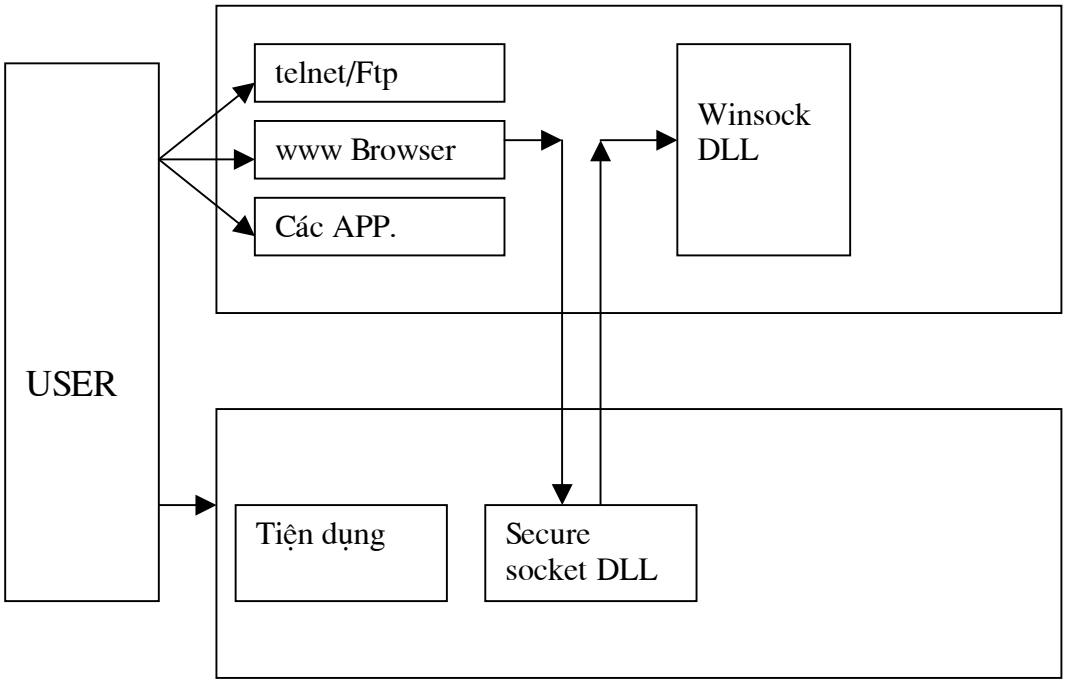
- Bất kỳ một thư viện liên kết động nào (.DLL) đều có thể đóng vai thư viện Winsock bằng việc xuất khẩu các tên hàm giống như Winsock. Do vậy đổi tên file Secure socket "Winsock.dll" và cho file Winsock.dll ban đầu một tên khác chẳng hạn "ORGsock.dll". Điều này cho phép Secure socket chặn lời gọi của một ứng dụng tới các hàm thư viện Winsock.

Phương pháp này không phụ thuộc vào hệ điều hành. Hình 3 minh hoạ phương pháp đổi tên để chặn. Sau khi chương trình ứng dụng đã được khởi sinh thì Secure socket DLL đã được đổi tên thành Winsock.dll sẽ được tải bởi chương trình Loader của hệ thống. Sau đó Secure socket DLL sẽ tải Winsock DLL ban đầu mà đã được đổi tên thành ORGsock.dll. Khi chương trình ứng dụng gọi hàm Winsock thì hàm tương ứng trong Secure socket DLL sẽ được gọi. Nó sẽ nén và mã hoá dữ liệu và gọi hàm trong Winsock DLL ban đầu.

3.2. Khung dữ liệu

Để hiệu quả và an toàn, các khối dữ liệu cần được mã và nén. Do vậy, Secure socket chia dòng dữ liệu thành nhiều frame, sau đó nén và mã chúng. Thứ tự là quan trọng bởi vì sau mã hoá dữ liệu là ngẫu nhiên và không nén được nữa. Frame có header đã được gắn xác định kiểu và độ lớn nội dung được truyền tới người nhận.

Secure socket nhận dòng dữ liệu từ TCP/IP và kiểm tra Header lắp vào Frame, sau đó giải mã, giải nén dữ liệu và chuyển tới ứng dụng. Hình 4 cho xem lược đồ khung dữ liệu.



Hình 2. Cấu trúc Secure socket chặn các lệnh của Winsock

3.3. Thao tác kiểu dị bộ

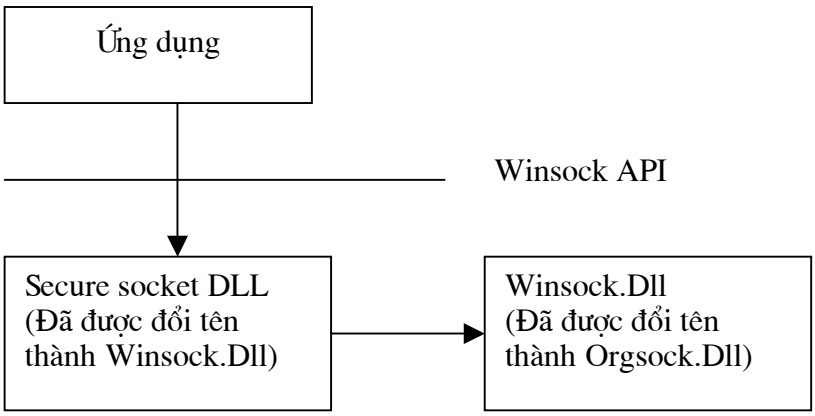
Khi sử dụng các hàm của Winsock, có hai dạng thao tác: Dạng đồng bộ và dạng dị bộ. Các hàm đồng bộ đợi đến khi các phép toán mạng đã yêu cầu được hoàn tất trước khi trả lại lời gọi hàm (lúc đó mới có thể gọi tiếp). Trong khi gọi hàm theo kiểu dị bộ trả lại ngay tức thì mà không quan tâm đến thao tác mạng đã được hoàn tất hay chưa. Khi thao tác được hoàn tất, Winsock gửi một thông báo tới chương trình ứng dụng để thông báo rằng thao tác còn đang treo đã hoàn tất. Trong trường hợp này, thông báo phải bị chặn lại.

Vì mục đích này, chúng tôi sử dụng hàm Winsock WSAAsynselect (hàm này được dùng để đăng ký hàm của Windows) để nhận thông báo và thay đổi Mode về dị bộ. Secure Socket chặn WSAAsynselect và thay thế tham số "Windows handle" của nó bằng "Windows handle" của Secure socket. Sau đó phát lại lệnh tới Winsock.Dll. Bởi vậy Secure socket có thể chặn thông báo từ Winsock.Dll, xử lý nó và nếu cần thiết gửi thông báo tới Windows ban đầu.

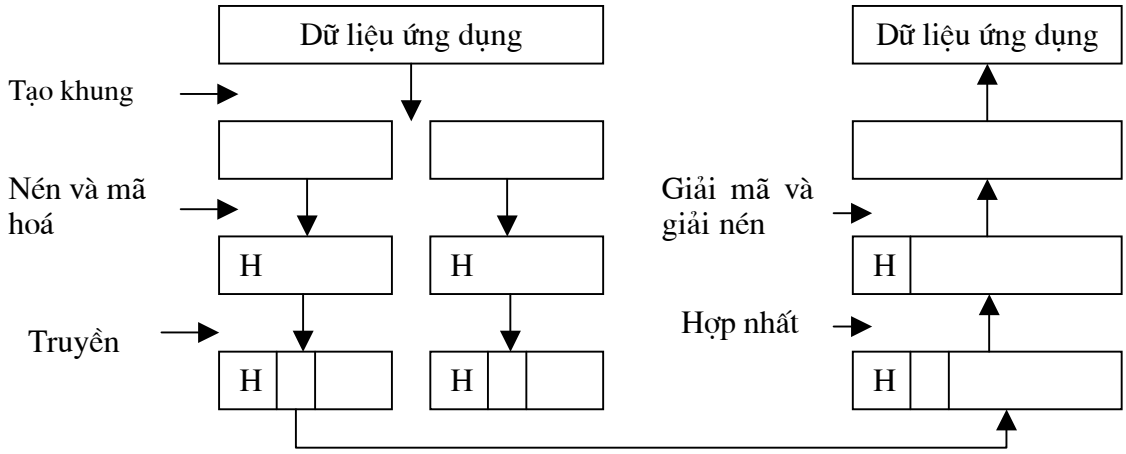
3.4. Thao tác cơ bản

Ở dạng dị bộ, hàm send() của Winsock ghi một phần dữ liệu (từ 1 byte đến độ dài được yêu cầu phụ thuộc vào sự sẵn sàng của buffer) và trả lại kích thước của phần ghi được cho ứng dụng. Việc truyền dữ liệu được đảm bảo bởi Winsock. Nhưng nếu Secure socket chặn hàm send() và thực hiện nén và mã hoá dữ liệu trong đơn vị frame đã xác định trước thì nó phải trả lại kích thước của frame cho ứng dụng vì những lý do sau:

- Nói chung khi một frame đã được xử lý thì nó không thể chia thành những phần nhỏ hơn.
- Một khi frame đã được xử lý, nó không thể đặt lại trạng thái ban đầu bởi vì các từ điển được sử dụng để nén tăng lên ở cả máy trạm và máy chủ.



Hình 3. Phương pháp đổi tên để chặn



Hình 4. Khung dữ liệu

Chính vì vậy khi Secure socket truyền hỏng frame thì nó sẽ giữ frame và truyền lại ở chế độ nền cho đến khi việc truyền hoàn tất.

4. Thoả thuận

Để thiết lập kết nối an toàn giữa PC ở xa và Server phải có sự thoả thuận giữa chúng trước khi truyền dữ liệu. Trong chuỗi thoả thuận, Secure socket xác nhận Secure socket ở phần kia đã được cài đặt hay chưa, chọn các phương pháp nén, mã hoá, trao đổi khoá mật mã và thực hiện xác thực.

4.1. Xác thực

Mục đích của việc xác thực là để bảo vệ các Server khỏi bị truy nhập trái phép bằng việc cho phép chúng khả năng định danh các USER đã được đăng ký. Có thể sử dụng mật khẩu và các thuật toán mật mã đối xứng để xác thực. Phương pháp sử dụng mật khẩu nói chung đã quen biết. Với phương pháp này thì USER là hợp pháp nếu mật khẩu bí mật đã được biết bởi USER đã đăng ký đã được khai báo với Server. Thuật toán mật mã đối xứng cho phép Server và USER xác nhận nhau khi cả hai có cùng khoá.

Secure socket lựa chọn phương pháp này vì khoá mã hoá dữ liệu có thể nhận được từ khoá bí mật chung.

4.2. Chuỗi thoả thuận

Trước khi bắt đầu truyền tin mật, Client và Server phải biết những khả năng chung là những gì chẳng hạn thuật toán nén và mã hoá bằng một chuỗi những thoả thuận. Để tránh buộc một ứng dụng phải làm điều này, Secure socket chặn các hàm connect() và accept() và thực hiện thoả thuận. Việc xác thực cũng được làm trong quá trình thoả thuận.

1. *Kiểm tra đăng ký USER*

   Client gửi tên USER tới Server. Server kiểm tra xem tên USER đã được đăng ký tại Server hay chưa và trả lại kết quả cho Client. Số hiệu phiên bản (*version*) được gửi đi để đảm bảo chắc chắn rằng Client và Server sử dụng các phiên bản phần mềm Secure socket tương thích.

2. *Lựa chọn thuật toán và xác thực Server*

   Client gửi một danh sách các thuật toán đã sẵn sàng và một số ngẫu nhiên R_a để xác thực Server. Server phúc đáp bằng số hiệu thuật toán đã được lựa chọn, R_a đã nhận và một số ngẫu nhiên mới R_b cùng với khoá phiên key1. Mọi dữ liệu được mã hoá bằng khoá chung. Khoá phiên key1 được sử dụng để mã hoá dữ liệu ứng dụng từ Server. Client sau đó giải mã Ra và Rb.

chương trình thử nghiệm

Phần này sẽ trình bầy những modul cơ bản phục vụ cho thử nghiệm tư tưởng thiết kế đã trình bầy trong phần trước. Những kỹ thuật bảo vệ trình bầy trong phần này chỉ nhằm mục đích khẳng định những ý tưởng thiết kế trong phần trước là hoàn toàn khả thi. Các giao thức hội thoại giữa client và server được thiết kế để nhằm khẳng định chúng tôi có thể chủ động thực hiện hội thoại giữa Client và Server theo bất kỳ giao thức an toàn nào.

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <string.h>
#include <io.h>
#include <winsock.h>
#include <winbase.h>
//#include <malloc.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>
#include <process.h>
#include <sys\stat.h>
#include <atalkwsh.h>
#include "sev.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//#pragma comment(lib, "wsock32.lib")
char trung[20];
// CONST DEFINITION
#define MY_PORT                         1111
#define AUTH_STRING                     "ABC"
#define OK                              "OK"
#define DEST_IP_ADDR        "192.168.0.1"
// END OF DEFINITION



/*struct _ADDRESS_LIST_ {
        unsigned long ulAddress;
        struct _ADDRESS_LIST_  *pNext;
        struct _ADDRESS_LIST_  *pPrev;
};*/

unsigned long pList[20]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

```
DWORD dwCount = 0;
BOOL bContinue = TRUE;
int j;

/* Function */
void DllExit();
BOOL StartThread();
BOOL DoAuthentication(SOCKADDR_IN *name);
void AddToList(unsigned long ulAddr);
BOOL Exist(unsigned long ulAddr);
unsigned long AddServerAddress();
BOOL bThreadStart = FALSE;
BOOL bServer = FALSE;
BOOL bFirstTime = TRUE;
SOCKADDR_IN sin;
unsigned long GetAddr (LPSTR szHost);

HANDLE ulThreadHandle;
SOCKET sockListen;
void abc(char *p){FILE *fp=fopen("c:\\z.txt","a+");fprintf(fp,"%s\n",p);fclose(fp);}
void abs(char *p){FILE *fp=fopen("c:\\zs.txt","a+");fprintf(fp,"%s\n",p);fclose(fp);}
void abr(char *p){FILE *fp=fopen("c:\\zr.txt","a+");fprintf(fp,"%s\n",p);fclose(fp);}
void abt(char *p){FILE *fp=fopen("c:\\zt.txt","a+");fprintf(fp,"%s\n",p);fclose(fp);}
void atm(char *p){FILE *fp=fopen("c:\\ztm.txt","a+");fprintf(fp,"%s\n",p);fclose(fp);}

BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{

        switch (dwReason)
        {
                case DLL_PROCESS_ATTACH:
                case DLL_THREAD_ATTACH:
                        dwCount++;
                        break;
                case DLL_THREAD_DETACH:
                case DLL_PROCESS_DETACH:
                        dwCount--;
                        if(dwCount == 0)
                        {
                                bContinue = FALSE;
                                //for (j=0;j<20;j++) pList[j]=0;
                          //    DllExit();
                        }

                        break;
   }
        return 1;   // ok
}

HMODULE hModule = NULL;
```

```
char aa[1000];FARPROC a;DWORD d;HANDLE winH;BOOL CN=TRUE;
char cKh[2][5];int khoa=2;
BOOL xacnhan1=FALSE;
BOOL xacnhan2=FALSE;
BOOL Orcl=FALSE;
BOOL lan1=TRUE;
int kdau=27;
int hesoA=0;
int hesoB=0;
struct protoent FAR * (__stdcall *getprotobynumber1)(int );
BOOL (__stdcall *AcceptEx1) (IN SOCKET ,IN SOCKET ,IN PVOID ,IN DWORD ,IN
DWORD ,IN DWORD ,OUT LPDWORD ,IN LPOVERLAPPED );
VOID (__stdcall *GetAcceptExSockaddrs1)(IN PVOID,IN DWORD ,IN DWORD ,IN DWORD
,OUT struct sockaddr **,OUT LPINT ,OUT struct sockaddr **,OUT LPINT );
int (__stdcall *recvfrom1) (SOCKET , char FAR * , int, int ,struct sockaddr FAR *, int FAR * );
HANDLE (__stdcall * WSAAsyncGetServByName1)(HWND , u_int ,const char FAR * ,const
char FAR * ,char FAR * , int );
int (__stdcall *getsockopt1)(SOCKET ,int ,int ,char * , int * );
u_short (__stdcall *ntohs1)(u_short );
struct hostent *  (__stdcall *gethostbyname1)(const char FAR *  );
int (__stdcall *getsockname1)(SOCKET ,struct sockaddr *,int * );
int (__stdcall *bind1)(SOCKET ,const struct sockaddr *,int );
u_long (__stdcall *htonl1)(u_long);
char * (__stdcall *inet_ntoa1)(struct in_addr);
int (__stdcall *WsControl1)(int ,int ,int ,int ,int ,int );
unsigned long (__stdcall *inet_addr1)(const char FAR * );
int (__stdcall *__WSAFDIsSet1)(SOCKET,fd_set FAR *);
int (__stdcall *WSAGetLastError1)();
int (__stdcall *recv1)(SOCKET ,char FAR * ,int ,int );
int (__stdcall *send1)(SOCKET ,const char * ,int ,int);
int (__stdcall *connect1)(SOCKET,const struct sockaddr *,int);
int (__stdcall *closesockinfo1)(int );
int (__stdcall *NPLoadNameSpaces1)(int ,int ,int );
int (__stdcall *closesocket1)(SOCKET );
int (__stdcall *select1)(int ,fd_set FAR *,fd_set FAR *,fd_set FAR *,const struct timeval FAR *
);
HANDLE  (__stdcall *WSAAsyncGetHostByName1)(HWND ,u_int ,const char FAR * , char
FAR * ,int );
int (__stdcall *ioctlsocket1)(SOCKET ,long ,u_long FAR *);
int (__stdcall *setsockopt1)(SOCKET ,int ,int ,const char * ,int );
int (__stdcall *WSAAsyncSelect1)(SOCKET,HWND ,u_int,long);
SOCKET (__stdcall *socket1)(int ,int,int);u_short (__stdcall *htons1)(u_short);
int       (__stdcall *WSAStartup1)(WORD,LPWSADATA);int        (__stdcall *WSACleanup1)();
int (__stdcall *listen1)(SOCKET , int );
int (__stdcall *gethostname1 )(char FAR * , int );
SOCKET (__stdcall *accept1) (SOCKET , struct sockaddr FAR *,int FAR *);
FARPROC (__stdcall  *WSASetBlockingHook1)(FARPROC );
int (__stdcall  *shutdown1)(SOCKET , int );
struct protoent FAR * (__stdcall *getprotobyname1)(const char FAR * );
struct servent FAR *(__stdcall *getservbyname1)(const char FAR * ,const char FAR *);
```

```
BOOL (__stdcall *WSAIsBlocking1)(void);
struct servent FAR * (__stdcall *getservbyport1)(int , const char FAR * );
struct hostent FAR * (__stdcall *gethostbyaddr1)(const char FAR * ,int , int );
void (__stdcall *WSASetLastError1)(int );
int (__stdcall *WSACancelBlockingCall1)(void);
int (__stdcall *getpeername1) (SOCKET , struct sockaddr FAR *,int FAR *);
u_long (__stdcall *ntohl1) (u_long );
int (__stdcall *sendto1) (SOCKET , const char FAR * buf, int len, int flag,const struct sockaddr
FAR *, int);
int (__stdcall *SetServiceA1) (
   IN    DWORD             ,
   IN    DWORD             ,
   IN    DWORD             ,
   IN    LPSERVICE_INFOA     ,
   IN    LPSERVICE_ASYNC_INFO ,
   IN OUT LPDWORD );
int (__stdcall *EnumProtocolsA1) (
   IN    LPINT  ,
   IN OUT LPVOID ,
   IN OUT LPDWORD );
int (__stdcall *GetTypeByNameA1) (
   IN    LPSTR ,
   IN OUT LPGUID );
int (__stdcall *GetAddressByNameA1) (
   IN    DWORD ,
   IN    LPGUID,
   IN    LPSTR ,
   IN    LPINT ,
   IN    DWORD ,
   IN    LPSERVICE_ASYNC_INFO ,
   IN OUT LPVOID ,
   IN OUT LPDWORD,
   IN OUT LPSTR ,
   IN OUT LPDWORD );
int (__stdcall *GetNameByTypeA1) (
   IN    LPGUID,
   IN OUT LPSTR,
   IN    DWORD );
int (__stdcall *GetServiceA1)(
   IN    DWORD,
   IN    LPGUID,
   IN    LPSTR,
   IN    DWORD,
   IN OUT LPVOID,
   IN OUT LPDWORD,
   IN    LPSERVICE_ASYNC_INFO );
BOOL (__stdcall *TransmitFile1 )(IN SOCKET ,
                                 IN HANDLE ,
                                 IN DWORD ,
                                 IN DWORD,
```

```
                                                        IN LPOVERLAPPED ,
                                                        IN
LPTRANSMIT_FILE_BUFFERS ,
                                                        IN DWORD );

int PASCAL FAR WSAStartup(WORD wVersionRequired, LPWSADATA lpWSAData)
{
        int nRes;

        if(hModule == NULL)
                hModule=LoadLibrary("wsock32.aaa");
        if(hModule == NULL)
        {
                ::MessageBox(NULL, "hModule = NULL", "Error", MB_OK);
                WSASetLastError(WSASYSNOTREADY);
                return SOCKET_ERROR;
        }
        a=GetProcAddress(hModule,"WSAStartup");

        WSAStartup1=(int (_stdcall *)(WORD,LPWSADATA))a;
        nRes = WSAStartup1(wVersionRequired,lpWSAData);
        return nRes;

}
int PASCAL FAR WSACleanup(void)
{

        a=GetProcAddress(hModule,"WSACleanup");
   WSACleanup1=(int (_stdcall *)())a;
        return WSACleanup1();
}
u_short PASCAL FAR htons (u_short hostshort)
{

        a=GetProcAddress(hModule,"htons");
        htons1=(u_short (_stdcall *)(u_short))a;
        return htons1(hostshort);
}
SOCKET PASCAL FAR socket (int af, int type, int protocol)
{

        a=GetProcAddress(hModule,"socket");
        socket1=(SOCKET (_stdcall *)(int ,int,int))a;
        return socket1(af,type,protocol);
}
int PASCAL FAR WSAAsyncSelect(SOCKET s, HWND hWnd, u_int wMsg,long lEvent)
{

        a=GetProcAddress(hModule,"WSAAsyncSelect");
        WSAAsyncSelect1=(int (_stdcall *)(SOCKET,HWND ,u_int,long ))a;
```

```
	return WSAAsyncSelect1(s,hWnd,wMsg,lEvent);
}
int PASCAL FAR setsockopt(SOCKET s,int level,int optname,const char * optval,int optlen)
{
	a=GetProcAddress(hModule,"setsockopt");
	setsockopt1=(int (_stdcall *)(SOCKET ,int ,int ,const char * ,int ))a;
	return setsockopt1(s,level,optname,optval,optlen);
}
int PASCAL FAR ioctlsocket(SOCKET s, long cmd, u_long FAR *argp)
{
	int io;
	a=GetProcAddress(hModule,"ioctlsocket");
	ioctlsocket1=(int (_stdcall *)(SOCKET ,long ,u_long FAR *))a;
	io=ioctlsocket1(s,cmd,argp);
	return io;
}
HANDLE PASCAL FAR WSAAsyncGetHostByName(HWND hWnd, u_int wMsg,const char
FAR * name, char FAR * buf,int buflen)
{
	a=GetProcAddress(hModule,"WSAAsyncGetHostByName");
	WSAAsyncGetHostByName1=(HANDLE (_stdcall *)(HWND ,u_int ,const char FAR * ,
char FAR * ,int ))a;
	return WSAAsyncGetHostByName1(hWnd,wMsg,name,buf,buflen);
}
int PASCAL FAR select(int nfds, fd_set FAR *readfds, fd_set FAR *writefds,fd_set FAR
*exceptfds, const struct timeval FAR *timeout)
{

	a=GetProcAddress(hModule,"select");
	select1=(int (_stdcall *)(int ,fd_set FAR *,fd_set FAR *,fd_set FAR *,const struct
timeval FAR *))a;
	return select1(nfds,readfds,writefds,exceptfds,timeout);
}
int PASCAL FAR recvfrom (SOCKET s, char FAR * buf, int len, int flags,struct sockaddr FAR
*from, int FAR * fromlen)
{
	int c;


	a=GetProcAddress(hModule,"recvfrom");
	recvfrom1=(int (_stdcall *)(SOCKET,char FAR *,int,int,struct sockaddr FAR *,int FAR
* ))a;
	c=recvfrom1(s,buf,len,flags,from,fromlen);
	abs(buf);
	return c;
}
int PASCAL FAR closesocket(SOCKET s)
{

	a=GetProcAddress(hModule,"closesocket");closesocket1=(int (_stdcall
```

```
                                        *)(SOCKET ))a;
        return closesocket1(s);
}
int PASCAL FAR NPLoadNameSpaces(int p,int q,int r)
{
        a=GetProcAddress(hModule,"NPLoadNameSpaces");
        NPLoadNameSpaces1=(int (_stdcall *)(int ,int ,int ))a;
        return NPLoadNameSpaces1(p,q,r);
}

int PASCAL FAR closesockinfo(int p)
{
  a=GetProcAddress(hModule,"closesockinfo");
  closesockinfo1=(int (_stdcall *)(int))a;
  return closesockinfo1(p);
}
int PASCAL FAR connect(SOCKET s,const struct sockaddr *name, int namelen)
{
        int n;
        a=GetProcAddress(hModule,"connect");
        connect1=(int (_stdcall *)(SOCKET ,const struct sockaddr *,int ))a;
        n = connect1(s, name, namelen);
        return n;
}
int PASCAL FAR WSAGetLastError(void)
{
        a=GetProcAddress(hModule,"WSAGetLastError");WSAGetLastError1=(int (_stdcall
*)())a;
        d=WSAGetLastError1();
        sprintf(aa,"WSAGetLastError= %d",d);
        return d;
}

int PASCAL FAR send(SOCKET s,const char  FAR * buf,int len,int flags)
{
        int nRes;

        idea_en_file((unsigned char *)trung,(unsigned char *)buf,len);
        a=GetProcAddress(hModule,"send");
         send1=(int (_stdcall *)(SOCKET ,const char FAR * ,int ,int ))a;
         nRes=send1(s,buf,len,flags);
        return nRes;
}

int PASCAL FAR recv(SOCKET s, char FAR * buf, int len, int flags)
{
        int c,x;
        int ii;
```

```
	len=2048;
	a=GetProcAddress(hModule,"recv");
	recv1=(int (_stdcall *)(SOCKET ,char FAR * ,int ,int ))a;
	c=recv1(s, buf, len, flags);

	if(c>0)
	{
		idea_de_file((unsigned char *)trung,(unsigned char *)buf,c);
	}
	return c;//recv1(s, buf, len, flags);
}

int PASCAL FAR __WSAFDIsSet(SOCKET p,fd_set FAR *q)
{

   a=GetProcAddress(hModule,"__WSAFDIsSet");
   __WSAFDIsSet1=(int (_stdcall *)(SOCKET,fd_set FAR *))a;
   return __WSAFDIsSet1(p,q);
}

unsigned long PASCAL FAR inet_addr(const char FAR * cp)
{
	a=GetProcAddress(hModule,"inet_addr");
	inet_addr1=(unsigned long (_stdcall *)(const char FAR * ))a;
	return inet_addr1(cp);
}

int PASCAL  FAR  WsControl(int p,int q,int r,int s,int t,int u)
{
	a=GetProcAddress(hModule,"WsControl");
	WsControl1=(int (_stdcall *)(int ,int ,int ,int ,int ,int ))a;
	return WsControl1(p,q,r,s,t,u);
}

char * PASCAL FAR inet_ntoa (struct in_addr in)
{
   a=GetProcAddress(hModule,"inet_ntoa");
   inet_ntoa1=(char *  (_stdcall *)(struct in_addr))a;
   return inet_ntoa1(in);
}

u_long PASCAL FAR htonl(u_long hostlong)
{

	a=GetProcAddress(hModule,"htonl");htonl1=(u_long  (_stdcall *)(u_long))a;
	return htonl1(hostlong);
}

int PASCAL bind(SOCKET s, const struct sockaddr FAR *addr, int namelen)
{
```

```
	a=GetProcAddress(hModule,"bind");
	bind1=(int (_stdcall *)(SOCKET ,const struct sockaddr *,int ))a;
	return bind1(s,addr,namelen);
}

int PASCAL getsockname(SOCKET s, struct sockaddr *name,int * namelen)
{
	a=GetProcAddress(hModule,"getsockname");
	getsockname1=(int (_stdcall *)(SOCKET ,struct sockaddr *,int * ))a;
	return getsockname1(s,name,namelen);
}

struct hostent * PASCAL FAR gethostbyname(const char FAR * name)
{
	a=GetProcAddress(hModule,"gethostbyname");
	gethostbyname1=(struct hostent * (_stdcall *)(const char FAR * ))a;
	return gethostbyname1(name);
}

u_short PASCAL ntohs(u_short netshort)
{
	a=GetProcAddress(hModule,"ntohs");
	ntohs1=(u_short (_stdcall *)(u_short))a;
	return ntohs1(netshort);
}

int PASCAL getsockopt(SOCKET s,int level,int optname,char * optval, int *optlen)
{

	a=GetProcAddress(hModule,"getsockopt");
	getsockopt1=(int (_stdcall *)(SOCKET ,int ,int ,char * , int *))a;
	return getsockopt1(s,level,optname,optval,optlen);
}

int PASCAL FAR listen (SOCKET s, int backlog)
{
	a=GetProcAddress(hModule,"listen");
	listen1=(int (_stdcall *)(SOCKET,int))a;
	return listen1(s,backlog);
}
int PASCAL FAR gethostname (char FAR * name, int namelen)
{
	a=GetProcAddress(hModule,"gethostname");
	gethostname1=(int (_stdcall *)(char FAR *,int))a;
	return gethostname1(name,namelen);
}

SOCKET PASCAL FAR accept (SOCKET s, struct sockaddr FAR *addr,int FAR *addrlen)
{
	SOCKET sockAccept;
```

```
	if( (! bThreadStart) && (bFirstTime) )
	{
		bFirstTime = FALSE;
		bServer = TRUE;
		if(StartThread())
			bThreadStart = TRUE;
			}

	a=GetProcAddress(hModule,"accept");

	accept1=(SOCKET (_stdcall *)(SOCKET,struct sockaddr FAR *,int FAR *))a;
	sockAccept = accept1(s,addr,addrlen);

	return sockAccept;

}

FARPROC PASCAL FAR WSASetBlockingHook(FARPROC pBlockFunc)
{

	a=GetProcAddress(hModule,"WSASetBlockingHook");
	WSASetBlockingHook1=(FARPROC (_stdcall *)(FARPROC))a;
	return WSASetBlockingHook1(lpBlockFunc);
}

int PASCAL FAR shutdown (SOCKET s, int how)
{
	a=GetProcAddress(hModule,"shutdown");
	shutdown1=(int (_stdcall *)(SOCKET,int))a;return shutdown1(s,how);
}
struct protoent FAR * PASCAL FAR getprotobyname(const char FAR * name)
{
	a=GetProcAddress(hModule,"getprotobyname");
	getprotobyname1=(struct protoent FAR * (_stdcall *)(const char FAR *))a;
	return getprotobyname1(name);
}

struct servent FAR * PASCAL FAR getservbyname(const char FAR * name,const char FAR *
proto)
{
	a=GetProcAddress(hModule,"getservbyname");
	getservbyname1=(struct servent FAR * (_stdcall *)(const char FAR *,const char FAR
*))a;
	return getservbyname1(name,proto);
}

BOOL PASCAL FAR WSAIsBlocking(void)
{
```

```
	a=GetProcAddress(hModule,"WSAIsBlocking");
	WSAIsBlocking1= (BOOL (_stdcall *)(void))a;
	return WSAIsBlocking1();
}

void PASCAL FAR WSASetLastError(int rError)
{

	 a=GetProcAddress(hModule,"WSASetLastError");
	 WSASetLastError1=(void (_stdcall *)(int))a;
	 WSASetLastError1(rError);
}

struct servent FAR * PASCAL FAR getservbyport(int port, const char FAR * proto)
{
	a=GetProcAddress(hModule,"getservbyport");
	getservbyport1=(struct servent FAR * (_stdcall *)(int,const char FAR *))a;
	return getservbyport1(port,proto);
}

struct hostent FAR * PASCAL FAR gethostbyaddr(const char FAR * addr,int len, int type)
{
	a=GetProcAddress(hModule,"gethostbyaddr");
	gethostbyaddr1=(struct hostent FAR * (_stdcall *)(const char FAR *,int,int))a;
	return gethostbyaddr1(addr,len,type);
}

int PASCAL FAR WSACancelBlockingCall(void)
{
	a=GetProcAddress(hModule,"WSACancelBlockingCall");
	WSACancelBlockingCall1=(int (_stdcall *)(void))a;
	return WSACancelBlockingCall1();
}

int PASCAL FAR SetServiceA (
		IN	DWORD		dwNameSpace,
		IN	DWORD		dwOperation,
		IN	DWORD		dwFlags,
		IN	LPSERVICE_INFOA	lpServiceInfo,
		IN	LPSERVICE_ASYNC_INFO lpServiceAsyncInfo,
		IN OUT  LPDWORD		lpdwStatusFlags)
{
	a=GetProcAddress(hModule,"SetServiceA");
	SetServiceA1=(int (_stdcall *)(IN  DWORD,IN  DWORD,IN  DWORD,IN
LPSERVICE_INFOA, IN	LPSERVICE_ASYNC_INFO, IN  OUT  LPDWORD ))a;
	return
SetServiceA1(dwNameSpace,dwOperation,dwFlags,lpServiceInfo,lpServiceAsyncInfo,lpdwStatus
Flags);
}
```

```
int PASCAL FAR EnumProtocolsA (
   IN     LPINT          lpiProtocols,
   IN OUT LPVOID         lpProtocolBuffer,
   IN OUT LPDWORD        lpdwBufferLength)
{
        a=GetProcAddress(hModule,"EnumProtocolsA");
        EnumProtocolsA1=(int (_stdcall *)(IN     LPINT,IN OUT LPVOID,IN OUT
LPDWORD))a;
        return EnumProtocolsA1(lpiProtocols,lpProtocolBuffer,lpdwBufferLength);
}

int PASCAL FAR GetTypeByNameA (
   IN     LPSTR        lpServiceName,
   IN OUT LPGUID         lpServiceType
   )
{
         a=GetProcAddress(hModule,"GetTypeByNameA");
        GetTypeByNameA1=(int (_stdcall *)(IN LPSTR, IN OUT LPGUID))a;
        return GetTypeByNameA1(lpServiceName,lpServiceType);
}

int PASCAL FAR GetAddressByNameA (
   IN     DWORD              dwNameSpace,
   IN     LPGUID             lpServiceType,
   IN     LPSTR            lpServiceName OPTIONAL,
   IN     LPINT             lpiProtocols OPTIONAL,
   IN     DWORD              dwResolution,
   IN     LPSERVICE_ASYNC_INFO lpServiceAsyncInfo OPTIONAL,
   IN OUT LPVOID             lpCsaddrBuffer,
   IN OUT LPDWORD              lpdwBufferLength,
   IN OUT LPSTR            lpAliasBuffer OPTIONAL,
   IN OUT LPDWORD              lpdwAliasBufferLength OPTIONAL
   )
{
        a=GetProcAddress(hModule,"GetAddressByNameA");
        GetAddressByNameA1=(int (_stdcall *)( IN     DWORD ,
                              IN     LPGUID,
                              IN     LPSTR ,
                              IN     LPINT ,
                              IN     DWORD ,
                              IN     LPSERVICE_ASYNC_INFO ,
                              IN OUT LPVOID ,
                              IN OUT LPDWORD,
                              IN OUT LPSTR ,
                              IN OUT LPDWORD))a;

        return GetAddressByNameA1( dwNameSpace,
                        lpServiceType,
                        lpServiceName OPTIONAL,
                        lpiProtocols OPTIONAL,
```

```
                    dwResolution,
                    lpServiceAsyncInfo OPTIONAL,
                    lpCsaddrBuffer,
                    lpdwBufferLength,
                    lpAliasBuffer OPTIONAL,
                    lpdwAliasBufferLength OPTIONAL);
}

int PASCAL FAR GetNameByTypeA (
   IN     LPGUID         lpServiceType,
   IN OUT LPSTR          lpServiceName,
   IN     DWORD          dwNameLength
   )
{
        a=GetProcAddress(hModule,"GetNameByTypeA");
         GetNameByTypeA1=(int (_stdcall *)(IN     LPGUID,IN OUT LPSTR,IN     DWORD ))a;
         return GetNameByTypeA1(lpServiceType,lpServiceName,dwNameLength);
}

int PASCAL FAR GetServiceA (
   IN     DWORD              dwNameSpace,
   IN     LPGUID             lpGuid,
   IN     LPSTR              lpServiceName,
   IN     DWORD              dwProperties,
   IN OUT LPVOID             lpBuffer,
   IN OUT LPDWORD            lpdwBufferSize,
   IN     LPSERVICE_ASYNC_INFO lpServiceAsyncInfo
   )
{
        a=GetProcAddress(hModule,"GetServiceA");
        GetServiceA1=(int (_stdcall *)(IN     DWORD,
                    IN     LPGUID,
                    IN     LPSTR,
                    IN     DWORD,
                    IN OUT LPVOID,
                    IN OUT LPDWORD,
                    IN     LPSERVICE_ASYNC_INFO ))a;
        return
GetServiceA1(dwNameSpace,lpGuid,lpServiceName,dwProperties,lpBuffer,lpdwBufferSize,lpSer
viceAsyncInfo);
}
BOOL PASCAL FAR TransmitFile (IN SOCKET hSocket,
                                                IN HANDLE hFile,
                                                IN DWORD nNumberOfBytesToWrite,
                                                IN DWORD nNumberOfBytesPerSend,
                                                IN LPOVERLAPPED lpOverlapped,
                                                IN LPTRANSMIT_FILE_BUFFERS
lpTransmitBuffers,
                                                IN DWORD dwReserved)
{
```

```
	// LPOFSTRUCT  lpOpenBuff;
	a=GetProcAddress(hModule,"TransmitFile");
	TransmitFile1=(BOOL (_stdcall *)(IN SOCKET,
				IN HANDLE ,
							IN DWORD ,
							IN DWORD ,
							IN LPOVERLAPPED ,
							IN
LPTRANSMIT_FILE_BUFFERS ,
							IN DWORD ))a;
	return TransmitFile1( hSocket, hFile, nNumberOfBytesToWrite,
nNumberOfBytesPerSend, lpOverlapped, lpTransmitBuffers, dwReserved);
}


int PASCAL FAR WEP(int p)
{
	return 1;
}

BOOL PASCAL FAR AcceptEx (IN SOCKET sListenSocket,IN SOCKET sAcceptSocket,IN
PVOID lpOutputBuffer,IN DWORD dwReceiveDataLength,IN DWORD
dwLocalAddressLength,IN DWORD dwRemoteAddressLength,OUT LPDWORD
lpdwBytesReceived,IN LPOVERLAPPED lpOverlapped)
{

	a=GetProcAddress(hModule,"AcceptEx");
	AcceptEx1=(BOOL (_stdcall *)(IN SOCKET ,IN SOCKET ,IN PVOID ,IN DWORD ,IN
DWORD ,IN DWORD ,OUT LPDWORD ,IN LPOVERLAPPED ))a;
	return AcceptEx1( sListenSocket, sAcceptSocket,lpOutputBuffer,dwReceiveDataLength,
	dwLocalAddressLength, dwRemoteAddressLength,lpdwBytesReceived, lpOverlapped);

}

VOID PASCAL FAR GetAcceptExSockaddrs (IN PVOID lpOutputBuffer,IN DWORD
dwReceiveDataLength,IN DWORD dwLocalAddressLength,IN DWORD
dwRemoteAddressLength,OUT struct sockaddr **LocalSockaddr,OUT LPINT
LocalSockaddrLength,OUT struct sockaddr **RemoteSockaddr,OUT LPINT
RemoteSockaddrLength)
{

	a=GetProcAddress(hModule,"GetAcceptExSockaddrs");
	GetAcceptExSockaddrs1=(void (_stdcall *)(IN PVOID,IN DWORD,IN DWORD,IN
DWORD ,OUT struct sockaddr **,OUT LPINT ,OUT struct sockaddr **,OUT LPINT ))a;
	GetAcceptExSockaddrs1(lpOutputBuffer,dwReceiveDataLength,dwLocalAddressLength,
	dwRemoteAddressLength, LocalSockaddr,
	LocalSockaddrLength,RemoteSockaddr,RemoteSockaddrLength);
}

int PASCAL FAR getpeername (SOCKET s, struct sockaddr FAR *name,int FAR * namelen)
```

```
{
	a=GetProcAddress(hModule,"getpeername");
	getpeername1=(int (_stdcall *)(SOCKET,struct sockaddr FAR *,int FAR *))a;
	khoa=0;
	return getpeername1(s,name,namelen);
}
u_long PASCAL FAR ntohl (u_long netlong)
{

	a=GetProcAddress(hModule,"ntohl");
	ntohl1=(u_long (_stdcall *)(u_long))a;
	return ntohl1(netlong);
}

int PASCAL FAR sendto (SOCKET s, const char FAR * buf, int len, int flags,const struct
sockaddr FAR *to, int tolen)
{
	a=GetProcAddress(hModule,"sendto");
	sendto1=(int (_stdcall *)(SOCKET,const char FAR *,int,int,const struct sockaddr FAR
*,int))a;
	return sendto1(s,buf,len,flags,to,tolen);
}
struct protoent FAR * PASCAL FAR getprotobynumber(int proto)
{

	a=GetProcAddress(hModule,"getprotobynumber");
	getprotobynumber1=(struct protoent FAR * (_stdcall *)(int))a;
	return getprotobynumber1(proto);
}
HANDLE PASCAL FAR WSAAsyncGetServByName(HWND hWnd, u_int wMsg,const char
FAR * name,const char FAR * proto,char FAR * buf, int buflen)
{

	a=GetProcAddress(hModule,"WSAAsyncGetServByName");
	WSAAsyncGetServByName1=(HANDLE (_stdcall *)(HWND,u_int,const char FAR
*,const char FAR *,char FAR *,int))a;
	return WSAAsyncGetServByName1(hWnd,wMsg,name,proto,buf,buflen);
}
HANDLE PASCAL FAR WSAAsyncGetServByPort(HWND hWnd, u_int wMsg, int port,const
char FAR * proto, char FAR * buf,int buflen)
{
	return 0;
}
HANDLE PASCAL FAR WSAAsyncGetProtoByName(HWND hWnd, u_int wMsg,const char
FAR * name, char FAR * buf,int buflen)
{
	return 0;
}
HANDLE PASCAL FAR WSAAsyncGetProtoByNumber(HWND hWnd, u_int wMsg,int
number, char FAR * buf,int buflen)
```

```
{
        return 0;
}
HANDLE PASCAL FAR WSAAsyncGetHostByAddr(HWND hWnd, u_int wMsg,const char
FAR * addr, int len, int type,char FAR * buf, int buflen)
{
        return 0;
}
int PASCAL FAR WSACancelAsyncRequest(HANDLE hAsyncTaskHandle)
{
        return 0;
}
int PASCAL FAR WSAUnhookBlockingHook(void)
{
        return 0;
}
int PASCAL FAR WSARecvEx (SOCKET s, char FAR * buf, int len, int FAR *flags)
{
        return 0;
}
int PASCAL FAR Arecv (){return 0;}
int PASCAL FAR Asend (){return 0;}
int PASCAL FAR WSHEnumProtocols(){return 0;}
int PASCAL FAR inet_network (){return 0;}
int PASCAL FAR getnetbyname (){return 0;}
int PASCAL FAR rcmd  (){return 0;}
int PASCAL FAR rexec (){return 0;}
int PASCAL FAR rresvport (){return 0;}
int PASCAL FAR sethostname (){return 0;}
int PASCAL FAR dn_expand (){return 0;}
int PASCAL FAR s_perror  (){return 0;}
int PASCAL FAR GetAddressByNameW (){return 0;}
int PASCAL FAR EnumProtocolsW (){return 0;}
int PASCAL FAR GetTypeByNameW (){return 0;}
int PASCAL FAR GetNameByTypeW (){return 0;}
int PASCAL FAR SetServiceW (){return 0;}
int PASCAL FAR GetServiceW (){return 0;}

VOID ListenThread(VOID *pParam)
{
        char buf[100];
        int nRes;
        SOCKET sockClient;
        //SOCKADDR_IN addr;
        int iAddrLen=sizeof(SOCKADDR_IN);

        nRes = listen (sockListen, 1);
        if(nRes != SOCKET_ERROR)
   {
                a=GetProcAddress(hModule,"accept");
```

```
		accept1=(SOCKET (_stdcall *)(SOCKET,struct sockaddr FAR *,int FAR *))a;
	while(bContinue)
	{

		sockClient = accept1 (sockListen, (struct sockaddr*)&sin, &iAddrLen);

		if(sockClient == SOCKET_ERROR)
		{
			int n = WSAGetLastError();
			// WSAENOTSOCK)
			continue;
		}

		while(1)
		{

			a=GetProcAddress(hModule,"recv");
			recv1=(int (_stdcall *)(SOCKET ,char FAR * ,int ,int ))a;
			nRes = recv1(sockClient, (char*)buf, 100, 0);
			if( (nRes == 0) || (nRes == SOCKET_ERROR) )
				break;
			buf[nRes] = 0;
			abt("Da nhan roi");
			abt(buf);
			if(strcmp((const char*)buf, AUTH_STRING) == 0)
			{

				abt("Gui tro lai");
				a=GetProcAddress(hModule,"send");
				send1=(int (_stdcall *)(SOCKET ,const char FAR * ,int ,int ))a;
				send1(sockClient, OK, sizeof(OK), 0);
				bContinue=false;
				break;
			}
		}
		closesocket(sockClient);
	}
	}else abc("No listen !");

}

unsigned long AddServerAddress()
{
	TCHAR lpszName[MAX_COMPUTERNAME_LENGTH+1];
	DWORD iNameLen;
	unsigned long ulAddress;
	struct hostent *pHost;
	DWORD dwRes;

	iNameLen = MAX_COMPUTERNAME_LENGTH + 1;
```

```
	GetComputerName(lpszName, &iNameLen);
	ulAddress = inet_addr (lpszName);
	if (INADDR_NONE == ulAddress) {
		pHost = gethostbyname (lpszName);
		if (NULL == pHost)
		{
			dwRes = GetLastError ();
			abc("WSASetLastError _A");
			return 0;
		}

		memcpy((char FAR *)&ulAddress, pHost->h_addr, pHost->h_length);
	}
	return ulAddress;
}

BOOL StartThread()
{
	TCHAR lpszName[MAX_COMPUTERNAME_LENGTH+1];
	DWORD iNameLen;
	unsigned long ulAddress;
	struct hostent *pHost;
	//SOCKADDR_IN sin;
	int nRes;

	if(hModule == NULL)
	hModule=LoadLibrary("wsock32.aaa");
	sockListen = socket (AF_INET, SOCK_STREAM, 0);
	if (sockListen == INVALID_SOCKET)
	{
		int n = WSAGetLastError();
		abc("WSASetLastError _s");
		if(n == WSANOTINITIALISED)
		{
			return TRUE;
		} else
		{
			abc("Failed to create listen socket during Dll startup");
			return(FALSE);
		}
	}

	iNameLen = MAX_COMPUTERNAME_LENGTH + 1;
	GetComputerName(lpszName, &iNameLen);
	ulAddress = inet_addr (lpszName);
	if (INADDR_NONE == ulAddress) {
		pHost = gethostbyname (lpszName);
		if (NULL == pHost)
		{
			nRes = GetLastError ();
```

```
                abc("WSASetLastError _G");
                return FALSE;
            }

            memcpy((char FAR *)&ulAddress, pHost->h_addr, pHost->h_length);
        }

        sin.sin_family = PF_INET;
        sin.sin_addr.s_addr = ulAddress;
        sin.sin_port = htons(MY_PORT);
        nRes = bind (sockListen, (LPSOCKADDR) &sin, sizeof (sin));
        if (SOCKET_ERROR == nRes)
        {
            int n = WSAGetLastError();
            abc("WSASetLastError _b");
            if( n == WSAEADDRINUSE )
            {
                closesocket(sockListen);
                return TRUE;
            } else
            {
                abc("bind failed during Dll startup");
                closesocket(sockListen);
                return(FALSE);
            }
        }

        bContinue = TRUE;
        ulThreadHandle = (HANDLE)_beginthread(ListenThread, 0, NULL);
        if(ulThreadHandle == (HANDLE)-1)
        {
            closesocket(sockListen);
            return FALSE;
        }
        return TRUE;
}

BOOL DoAuthentication(SOCKADDR_IN *name)
{
        TCHAR lpszBuffer[40];
        SOCKET sockServer;
        SOCKADDR_IN sin;

        sockServer = socket (AF_INET, SOCK_STREAM, 0);
        if (INVALID_SOCKET == sockServer)
        {
            return(FALSE);
        }

        sin.sin_family = AF_INET;
```

```
	sin.sin_addr.s_addr = name->sin_addr.S_un.S_addr;
	sin.sin_port = htons (MY_PORT);

	a=GetProcAddress(hModule,"connect");
	connect1=(int (_stdcall *)(SOCKET ,const struct sockaddr *,int ))a;

	if( connect1(sockServer, (LPSOCKADDR) &sin, sizeof (sin)) == SOCKET_ERROR)
	{
		int iErr = WSAGetLastError();
		abc("connect failed");
		closesocket (sockServer);
		return(FALSE);
	}

	sprintf(lpszBuffer, "%s", AUTH_STRING);
	int n, iRes;
	n = strlen(lpszBuffer);
	iRes = send(sockServer, (const char*)lpszBuffer, n, 0);
	if(n == SOCKET_ERROR)
	{
		n = WSAGetLastError();
	} else  if(n != iRes)
	{
		closesocket(sockServer);
		return FALSE;
	}

	n = recv(sockServer, lpszBuffer, 30, 0);
	if(n == SOCKET_ERROR)
	{
		closesocket(sockServer);
		return FALSE;
	}
	closesocket(sockServer);

	lpszBuffer[n] = 0;
	abc(lpszBuffer);
	if(strcmp(lpszBuffer, OK) != 0) return FALSE;
	return TRUE;
}

BOOL Exist(unsigned long ulAddr)
{
 int j;
 for (j=0;j<20;j++)
	 if (pList[j]==ulAddr) return TRUE;
	 return FALSE;
}

void AddToList(unsigned long ulAddr)
```

```
{

          int j;

          if(Exist(ulAddr)) return;
         for (j=0;j<20 && pList[j]!=0 ;j++);
                  if (j<20) pList[j]=ulAddr;
}
unsigned long GetAddr (LPSTR szHost)
 {
  LPHOSTENT lpstHost;
  unsigned long lAddr = INADDR_ANY;

if (*szHost) {

lAddr = inet_addr (szHost);

if (lAddr == INADDR_NONE)
       {

     lpstHost = gethostbyname(szHost);
     if (lpstHost) {
       lAddr = *((unsigned long FAR *) (lpstHost->h_addr));
     } else {
       lAddr = INADDR_ANY;
     }
    }
  }
  return (lAddr);
}

#include <string.h>
#include <stdio.h>
#include <io.h>
#include <conio.h>
#include <stdlib.h>
#include "sev.h"
void mdstr(unsigned char s[255],byte *digest)
{
          MD5_CTX ctx;
          MD5Init(&ctx);
          MD5Update(&ctx,s,sizeof(s));
          MD5Final(digest, &ctx);
}
void byteReverse(unsigned char *buf, unsigned longs)
{
   uint32 t;
   do {
          t = (uint32) ((unsigned) buf[3] << 8 | buf[2]) << 16 |
             ((unsigned) buf[1] << 8 | buf[0]);
```

```
        *(uint32 *) buf = t;
        buf += 4;
    } while (--longs);
}
void MD5Init(MD5_CTX *ctx)
{
    ctx->buf[0] = 0x67452301;
    ctx->buf[1] = 0xefcdab89;
    ctx->buf[2] = 0x98badcfe;
    ctx->buf[3] = 0x10325476;

    ctx->bits[0] = 0;
    ctx->bits[1] = 0;
}

void MD5Update(struct MD5Context *ctx, unsigned char const *buf, unsigned len)
{
    uint32 t;

    t = ctx->bits[0];
    if ((ctx->bits[0] = t + ((uint32) len << 3)) < t)
        ctx->bits[1]++;
    ctx->bits[1] += len >> 29;

    t = (t >> 3) & 0x3f;
    if (t) {
        unsigned char *p = (unsigned char *) ctx->in + t;

        t = 64 - t;
        if (len < t) {
            memcpy(p, buf, len);
            return;
        }
        memcpy(p, buf, t);
        byteReverse(ctx->in, 16);
        MD5Transform(ctx->buf, (uint32 *) ctx->in);
        buf += t;
        len -= t;
    }
    while (len >= 64) {
        memcpy(ctx->in, buf, 64);
        byteReverse(ctx->in, 16);
        MD5Transform(ctx->buf, (uint32 *) ctx->in);
        buf += 64;
        len -= 64;
    }
    memcpy(ctx->in, buf, len);
}

void MD5Final(unsigned char digest[16], struct MD5Context *ctx)
```

```
{
   unsigned count;
   unsigned char *p;
   count = (ctx->bits[0] >> 3) & 0x3F;
   p = ctx->in + count;
   *p++ = 0x80;

   count = 64 - 1 - count;

   if (count < 8) {
         memset(p, 0, count);
         byteReverse(ctx->in, 16);
         MD5Transform(ctx->buf, (uint32 *) ctx->in);

         memset(ctx->in, 0, 56);
   } else {
         memset(p, 0, count - 8);
   }
   byteReverse(ctx->in, 14);

   ((uint32 *) ctx->in)[14] = ctx->bits[0];
   ((uint32 *) ctx->in)[15] = ctx->bits[1];

   MD5Transform(ctx->buf, (uint32 *) ctx->in);
   byteReverse((unsigned char *) ctx->buf, 4);
   memcpy(digest, ctx->buf, 16);
   memset(ctx, 0, sizeof(ctx));
}

#ifndef ASM_MD5

#define F1(x, y, z) (z ^ (x & (y ^ z)))
#define F2(x, y, z) F1(z, x, y)
#define F3(x, y, z) (x ^ y ^ z)
#define F4(x, y, z) (y ^ (x | ~z))

#ifdef __PUREC__
#define MD5STEP(f, w, x, y, z, data, s) \
         ( w += f+ data,  w = w<<s | w>>(32-s),  w += x )
#else
#define MD5STEP(f, w, x, y, z, data, s) \
         ( w += f(x, y, z) + data,  w = w<<s | w>>(32-s),  w += x )
#endif

void MD5Transform(uint32 buf[4], uint32 const in[16])
{
   register uint32 a, b, c, d;

   a = buf[0];
   b = buf[1];
```

```
    c = buf[2];
    d = buf[3];

#ifdef __PUREC__
    MD5STEP(F1(b,c,d), a, b, c, d, in[0] + 0xd76aa478L, 7);
    MD5STEP(F1(a,b,c), d, a, b, c, in[1] + 0xe8c7b756L, 12);
    MD5STEP(F1(d,a,b), c, d, a, b, in[2] + 0x242070dbL, 17);
    MD5STEP(F1(c,d,a), b, c, d, a, in[3] + 0xc1bdceeeL, 22);
    MD5STEP(F1(b,c,d), a, b, c, d, in[4] + 0xf57c0fafL, 7);
    MD5STEP(F1(a,b,c), d, a, b, c, in[5] + 0x4787c62aL, 12);
    MD5STEP(F1(d,a,b), c, d, a, b, in[6] + 0xa8304613L, 17);
    MD5STEP(F1(c,d,a), b, c, d, a, in[7] + 0xfd469501L, 22);
    MD5STEP(F1(b,c,d), a, b, c, d, in[8] + 0x698098d8L, 7);
    MD5STEP(F1(a,b,c), d, a, b, c, in[9] + 0x8b44f7afL, 12);
    MD5STEP(F1(d,a,b), c, d, a, b, in[10] + 0xffff5bb1L, 17);
    MD5STEP(F1(c,d,a), b, c, d, a, in[11] + 0x895cd7beL, 22);
    MD5STEP(F1(b,c,d), a, b, c, d, in[12] + 0x6b901122L, 7);
    MD5STEP(F1(a,b,c), d, a, b, c, in[13] + 0xfd987193L, 12);
    MD5STEP(F1(d,a,b), c, d, a, b, in[14] + 0xa679438eL, 17);
    MD5STEP(F1(c,d,a), b, c, d, a, in[15] + 0x49b40821L, 22);

    MD5STEP(F2(b,c,d), a, b, c, d, in[1] + 0xf61e2562L, 5);
    MD5STEP(F2(a,b,c), d, a, b, c, in[6] + 0xc040b340L, 9);
    MD5STEP(F2(d,a,b), c, d, a, b, in[11] + 0x265e5a51L, 14);
    MD5STEP(F2(c,d,a), b, c, d, a, in[0] + 0xe9b6c7aaL, 20);
    MD5STEP(F2(b,c,d), a, b, c, d, in[5] + 0xd62f105dL, 5);
    MD5STEP(F2(a,b,c), d, a, b, c, in[10] + 0x02441453L, 9);
    MD5STEP(F2(d,a,b), c, d, a, b, in[15] + 0xd8a1e681L, 14);
    MD5STEP(F2(c,d,a), b, c, d, a, in[4] + 0xe7d3fbc8L, 20);
    MD5STEP(F2(b,c,d), a, b, c, d, in[9] + 0x21e1cde6L, 5);
    MD5STEP(F2(a,b,c), d, a, b, c, in[14] + 0xc33707d6L, 9);
    MD5STEP(F2(d,a,b), c, d, a, b, in[3] + 0xf4d50d87L, 14);
    MD5STEP(F2(c,d,a), b, c, d, a, in[8] + 0x455a14edL, 20);
    MD5STEP(F2(b,c,d), a, b, c, d, in[13] + 0xa9e3e905L, 5);
    MD5STEP(F2(a,b,c), d, a, b, c, in[2] + 0xfcefa3f8L, 9);
    MD5STEP(F2(d,a,b), c, d, a, b, in[7] + 0x676f02d9L, 14);
    MD5STEP(F2(c,d,a), b, c, d, a, in[12] + 0x8d2a4c8aL, 20);

    MD5STEP(F3(b,c,d), a, b, c, d, in[5] + 0xfffa3942L, 4);
    MD5STEP(F3(a,b,c), d, a, b, c, in[8] + 0x8771f681L, 11);
    MD5STEP(F3(d,a,b), c, d, a, b, in[11] + 0x6d9d6122L, 16);
    MD5STEP(F3(c,d,a), b, c, d, a, in[14] + 0xfde5380cL, 23);
    MD5STEP(F3(b,c,d), a, b, c, d, in[1] + 0xa4beea44L, 4);
    MD5STEP(F3(a,b,c), d, a, b, c, in[4] + 0x4bdecfa9L, 11);
    MD5STEP(F3(d,a,b), c, d, a, b, in[7] + 0xf6bb4b60L, 16);
    MD5STEP(F3(c,d,a), b, c, d, a, in[10] + 0xbebfbc70L, 23);
    MD5STEP(F3(b,c,d), a, b, c, d, in[13] + 0x289b7ec6L, 4);
    MD5STEP(F3(a,b,c), d, a, b, c, in[0] + 0xeaa127faL, 11);
    MD5STEP(F3(d,a,b), c, d, a, b, in[3] + 0xd4ef3085L, 16);
    MD5STEP(F3(c,d,a), b, c, d, a, in[6] + 0x04881d05L, 23);
```

```
    MD5STEP(F3(b,c,d), a, b, c, d, in[9] + 0xd9d4d039L, 4);
    MD5STEP(F3(a,b,c), d, a, b, c, in[12] + 0xe6db99e5L, 11);
    MD5STEP(F3(d,a,b), c, d, a, b, in[15] + 0x1fa27cf8L, 16);
    MD5STEP(F3(c,d,a), b, c, d, a, in[2] + 0xc4ac5665L, 23);

    MD5STEP(F4(b,c,d), a, b, c, d, in[0] + 0xf4292244L, 6);
    MD5STEP(F4(a,b,c), d, a, b, c, in[7] + 0x432aff97L, 10);
    MD5STEP(F4(d,a,b), c, d, a, b, in[14] + 0xab9423a7L, 15);
    MD5STEP(F4(c,d,a), b, c, d, a, in[5] + 0xfc93a039L, 21);
    MD5STEP(F4(b,c,d), a, b, c, d, in[12] + 0x655b59c3L, 6);
    MD5STEP(F4(a,b,c), d, a, b, c, in[3] + 0x8f0ccc92L, 10);
    MD5STEP(F4(d,a,b), c, d, a, b, in[10] + 0xffeff47dL, 15);
    MD5STEP(F4(c,d,a), b, c, d, a, in[1] + 0x85845dd1L, 21);
    MD5STEP(F4(b,c,d), a, b, c, d, in[8] + 0x6fa87e4fL, 6);
    MD5STEP(F4(a,b,c), d, a, b, c, in[15] + 0xfe2ce6e0L, 10);
    MD5STEP(F4(d,a,b), c, d, a, b, in[6] + 0xa3014314L, 15);
    MD5STEP(F4(c,d,a), b, c, d, a, in[13] + 0x4e0811a1L, 21);
    MD5STEP(F4(b,c,d), a, b, c, d, in[4] + 0xf7537e82L, 6);
    MD5STEP(F4(a,b,c), d, a, b, c, in[11] + 0xbd3af235L, 10);
    MD5STEP(F4(d,a,b), c, d, a, b, in[2] + 0x2ad7d2bbL, 15);
    MD5STEP(F4(c,d,a), b, c, d, a, in[9] + 0xeb86d391L, 21);
#else
    MD5STEP(F1, a, b, c, d, in[0] + 0xd76aa478, 7);
    MD5STEP(F1, d, a, b, c, in[1] + 0xe8c7b756, 12);
    MD5STEP(F1, c, d, a, b, in[2] + 0x242070db, 17);
    MD5STEP(F1, b, c, d, a, in[3] + 0xc1bdceee, 22);
    MD5STEP(F1, a, b, c, d, in[4] + 0xf57c0faf, 7);
    MD5STEP(F1, d, a, b, c, in[5] + 0x4787c62a, 12);
    MD5STEP(F1, c, d, a, b, in[6] + 0xa8304613, 17);
    MD5STEP(F1, b, c, d, a, in[7] + 0xfd469501, 22);
    MD5STEP(F1, a, b, c, d, in[8] + 0x698098d8, 7);
    MD5STEP(F1, d, a, b, c, in[9] + 0x8b44f7af, 12);
    MD5STEP(F1, c, d, a, b, in[10] + 0xffff5bb1, 17);
    MD5STEP(F1, b, c, d, a, in[11] + 0x895cd7be, 22);
    MD5STEP(F1, a, b, c, d, in[12] + 0x6b901122, 7);
    MD5STEP(F1, d, a, b, c, in[13] + 0xfd987193, 12);
    MD5STEP(F1, c, d, a, b, in[14] + 0xa679438e, 17);
    MD5STEP(F1, b, c, d, a, in[15] + 0x49b40821, 22);

    MD5STEP(F2, a, b, c, d, in[1] + 0xf61e2562, 5);
    MD5STEP(F2, d, a, b, c, in[6] + 0xc040b340, 9);
    MD5STEP(F2, c, d, a, b, in[11] + 0x265e5a51, 14);
    MD5STEP(F2, b, c, d, a, in[0] + 0xe9b6c7aa, 20);
    MD5STEP(F2, a, b, c, d, in[5] + 0xd62f105d, 5);
    MD5STEP(F2, d, a, b, c, in[10] + 0x02441453, 9);
    MD5STEP(F2, c, d, a, b, in[15] + 0xd8a1e681, 14);
    MD5STEP(F2, b, c, d, a, in[4] + 0xe7d3fbc8, 20);
    MD5STEP(F2, a, b, c, d, in[9] + 0x21e1cde6, 5);
    MD5STEP(F2, d, a, b, c, in[14] + 0xc33707d6, 9);
    MD5STEP(F2, c, d, a, b, in[3] + 0xf4d50d87, 14);
```

```
    MD5STEP(F2, b, c, d, a, in[8] + 0x455a14ed, 20);
    MD5STEP(F2, a, b, c, d, in[13] + 0xa9e3e905, 5);
    MD5STEP(F2, d, a, b, c, in[2] + 0xfcefa3f8, 9);
    MD5STEP(F2, c, d, a, b, in[7] + 0x676f02d9, 14);
    MD5STEP(F2, b, c, d, a, in[12] + 0x8d2a4c8a, 20);

    MD5STEP(F3, a, b, c, d, in[5] + 0xfffa3942, 4);
    MD5STEP(F3, d, a, b, c, in[8] + 0x8771f681, 11);
    MD5STEP(F3, c, d, a, b, in[11] + 0x6d9d6122, 16);
    MD5STEP(F3, b, c, d, a, in[14] + 0xfde5380c, 23);
    MD5STEP(F3, a, b, c, d, in[1] + 0xa4beea44, 4);
    MD5STEP(F3, d, a, b, c, in[4] + 0x4bdecfa9, 11);
    MD5STEP(F3, c, d, a, b, in[7] + 0xf6bb4b60, 16);
    MD5STEP(F3, b, c, d, a, in[10] + 0xbebfbc70, 23);
    MD5STEP(F3, a, b, c, d, in[13] + 0x289b7ec6, 4);
    MD5STEP(F3, d, a, b, c, in[0] + 0xeaa127fa, 11);
    MD5STEP(F3, c, d, a, b, in[3] + 0xd4ef3085, 16);
    MD5STEP(F3, b, c, d, a, in[6] + 0x04881d05, 23);
    MD5STEP(F3, a, b, c, d, in[9] + 0xd9d4d039, 4);
    MD5STEP(F3, d, a, b, c, in[12] + 0xe6db99e5, 11);
    MD5STEP(F3, c, d, a, b, in[15] + 0x1fa27cf8, 16);
    MD5STEP(F3, b, c, d, a, in[2] + 0xc4ac5665, 23);

    MD5STEP(F4, a, b, c, d, in[0] + 0xf4292244, 6);
    MD5STEP(F4, d, a, b, c, in[7] + 0x432aff97, 10);
    MD5STEP(F4, c, d, a, b, in[14] + 0xab9423a7, 15);
    MD5STEP(F4, b, c, d, a, in[5] + 0xfc93a039, 21);
    MD5STEP(F4, a, b, c, d, in[12] + 0x655b59c3, 6);
    MD5STEP(F4, d, a, b, c, in[3] + 0x8f0ccc92, 10);
    MD5STEP(F4, c, d, a, b, in[10] + 0xffeff47d, 15);
    MD5STEP(F4, b, c, d, a, in[1] + 0x85845dd1, 21);
    MD5STEP(F4, a, b, c, d, in[8] + 0x6fa87e4f, 6);
    MD5STEP(F4, d, a, b, c, in[15] + 0xfe2ce6e0, 10);
    MD5STEP(F4, c, d, a, b, in[6] + 0xa3014314, 15);
    MD5STEP(F4, b, c, d, a, in[13] + 0x4e0811a1, 21);
    MD5STEP(F4, a, b, c, d, in[4] + 0xf7537e82, 6);
    MD5STEP(F4, d, a, b, c, in[11] + 0xbd3af235, 10);
    MD5STEP(F4, c, d, a, b, in[2] + 0x2ad7d2bb, 15);
    MD5STEP(F4, b, c, d, a, in[9] + 0xeb86d391, 21);
#endif

    buf[0] += a;
    buf[1] += b;
    buf[2] += c;
    buf[3] += d;
}

#endif

static uint16 mul(register uint16 a, register uint16 b)
```

```
{
    register word32 p;

    p = (word32) a *b;
    if (p) {
        b = low16(p);
        a = p >> 16;
        return (b - a) + (b < a);
    } else if (a) {
        return 1 - a;
    } else {
        return 1 - b;
    }
}
static uint16 mulInv(uint16 x)
{
    uint16 t0, t1;
    uint16 q, y;

    if (x <= 1)
        return x;
    t1 = 0x10001L / x;
    y = 0x10001L % x;
    if (y == 1)
        return low16(1 - t1);
    t0 = 1;
    do {
        q = x / y;
        x = x % y;
        t0 += q * t1;
        if (x == 1)
            return t0;
        q = y / x;
        y = y % x;
        t1 += q * t0;
    } while (y != 1);
    return low16(1 - t1);
}

static void ideaExpandKey(byte const *userkey, word16 * EK)
{
    int i, j;

    for (j = 0; j < 8; j++) {
        EK[j] = (userkey[0] << 8) + userkey[1];
        userkey += 2;
    }
    for (i = 0; j < IDEAKEYLEN; j++) {
        i++;
        EK[i + 7] = EK[i & 7] << 9 | EK[i + 1 & 7] >> 7;
```

```
            EK += i & 8;
            i &= 7;
        }
    }

    static void ideaInvertKey(word16 const *EK, word16 DK[IDEAKEYLEN])
    {
        int i;
        uint16 t1, t2, t3;
        word16 temp[IDEAKEYLEN];
        word16 *p = temp + IDEAKEYLEN;

        t1 = mulInv(*EK++);
        t2 = -*EK++;
        t3 = -*EK++;
        *--p = mulInv(*EK++);
        *--p = t3;
        *--p = t2;
        *--p = t1;

        for (i = 0; i < IDEAROUNDS - 1; i++) {
            t1 = *EK++;
            *--p = *EK++;
            *--p = t1;

            t1 = mulInv(*EK++);
            t2 = -*EK++;
            t3 = -*EK++;
            *--p = mulInv(*EK++);
            *--p = t2;
            *--p = t3;
            *--p = t1;
        }
        t1 = *EK++;
        *--p = *EK++;
        *--p = t1;

        t1 = mulInv(*EK++);
        t2 = -*EK++;
        t3 = -*EK++;
        *--p = mulInv(*EK++);
        *--p = t3;
        *--p = t2;
        *--p = t1;
        memcpy(DK, temp, sizeof(temp));
        burn(temp);
    }


    #ifndef USE68ASM
```

```
#define MUL(x,y) (x = mul(low16(x),y))
static void ideaCipher(byte const inbuf[8], byte outbuf[8],
                       word16 const *key)
{
    register uint16 x1, x2, x3, x4, s2, s3;
    word16 *in, *out;
    int r = IDEAROUNDS;

    in = (word16 *) inbuf;
    x1 = *in++;
    x2 = *in++;
    x3 = *in++;
    x4 = *in;
#ifndef HIGHFIRST
    x1 = (x1 >> 8) | (x1 << 8);
    x2 = (x2 >> 8) | (x2 << 8);
    x3 = (x3 >> 8) | (x3 << 8);
    x4 = (x4 >> 8) | (x4 << 8);
#endif
    do {
            MUL(x1, *key++);
            x2 += *key++;
            x3 += *key++;
            MUL(x4, *key++);

            s3 = x3;
            x3 ^= x1;
            MUL(x3, *key++);
            s2 = x2;
            x2 ^= x4;
            x2 += x3;
            MUL(x2, *key++);
            x3 += x2;

            x1 ^= x2;
            x4 ^= x3;

            x2 ^= s3;
            x3 ^= s2;
    } while (--r);
    MUL(x1, *key++);
    x3 += *key++;
    x2 += *key++;
    MUL(x4, *key);

    out = (word16 *) outbuf;
#ifdef HIGHFIRST
    *out++ = x1;
    *out++ = x3;
    *out++ = x2;
```

```
    *out = x4;
#else
    x1 = low16(x1);
    x2 = low16(x2);
    x3 = low16(x3);
    x4 = low16(x4);
    *out++ = (x1 >> 8) | (x1 << 8);
    *out++ = (x3 >> 8) | (x3 << 8);
    *out++ = (x2 >> 8) | (x2 << 8);
    *out = (x4 >> 8) | (x4 << 8);
#endif
}
#endif


void ideaCfbReinit(struct IdeaCfbContext *context, byte const *iv)
{
    if (iv)
        memcpy(context->iv, iv, 8);
    else
        fill0(context->iv, 8);
    context->bufleft = 0;
}

void ideaCfbInit(struct IdeaCfbContext *context, byte const key[16])
{
    ideaExpandKey(key, context->key);
    ideaCfbReinit(context, 0);
}

void ideaCfbDestroy(struct IdeaCfbContext *context)
{
    burn(*context);
}

void ideaCfbSync(struct IdeaCfbContext *context)
{
    int bufleft = context->bufleft;

    if (bufleft) {
        memmove(context->iv + bufleft, context->iv, 8 - bufleft);
        memcpy(context->iv, context->oldcipher + 8 - bufleft, bufleft);
        context->bufleft = 0;
    }
}


void ideaCfbEncrypt(struct IdeaCfbContext *context, byte const *src,
                    byte * dest, int count)
{
```

```
    int bufleft = context->bufleft;
    byte *bufptr = context->iv + 8 - bufleft;

if (count <= bufleft) {
            context->bufleft = bufleft - count;
            while (count--) {
                *dest++ = *bufptr++ ^= *src++;
            }
            return;
    }
    count -= bufleft;
while (bufleft--) {
            *dest++ = (*bufptr++ ^= *src++);
    }
while (count > 8) {
            bufptr = context->iv;
            memcpy(context->oldcipher, bufptr, 8);
            ideaCipher(bufptr, bufptr, context->key);
            bufleft = 8;
            count -= 8;
            do {
                *dest++ = (*bufptr++ ^= *src++);
            } while (--bufleft);
    }
    bufptr = context->iv;
    memcpy(context->oldcipher, bufptr, 8);
    ideaCipher(bufptr, bufptr, context->key);
    context->bufleft = 8 - count;
    do {
            *dest++ = (*bufptr++ ^= *src++);
    } while (--count);
}


void ideaCfbDecrypt(struct IdeaCfbContext *context, byte const *src,
                        byte * dest, int count)
{
    int bufleft = context->bufleft;
    static byte *bufptr;
    byte t;

    bufptr = context->iv + (8 - bufleft);
    if (count <= bufleft) {
            context->bufleft = bufleft - count;
            while (count--) {
                t = *bufptr;
                *dest++ = t ^ (*bufptr++ = *src++);
            }
            return;
    }
```

```
    count -= bufleft;
    while (bufleft--) {
        t = *bufptr;
        *dest++ = t ^ (*bufptr++ = *src++);
    }
    while (count > 8) {
        bufptr = context->iv;
        memcpy(context->oldcipher, bufptr, 8);
        ideaCipher(bufptr, bufptr, context->key);
        bufleft = 8;
        count -= 8;
        do {
            t = *bufptr;
            *dest++ = t ^ (*bufptr++ = *src++);
        } while (--bufleft);
    }
    bufptr = context->iv;
    memcpy(context->oldcipher, bufptr, 8);
    ideaCipher(bufptr, bufptr, context->key);
    context->bufleft = 8 - count;
    do {
        t = *bufptr;
        *dest++ = t ^ (*bufptr++ = *src++);
    } while (--count);
}

int  idea_en_file(unsigned char *pw,unsigned char *str,unsigned int lenstr)
{
        int status = 0;
        byte textbuf[5000],ideakey[24];
        struct IdeaCfbContext cfb;
        memcpy(textbuf,str,lenstr);
        mdstr(pw,ideakey);
        ideaCfbInit(&cfb, ideakey);
        ideaCfbSync(&cfb);
        ideaCfbEncrypt(&cfb, textbuf, textbuf, lenstr);
        ideaCfbDestroy(&cfb);
        memcpy(str,textbuf,lenstr);
        burn(textbuf);
        return status;
}

int  idea_de_file(unsigned char *pw,unsigned char *str,unsigned int lenstr)
{
        int status = 0;
        byte textbuf[5000],ideakey[16];
        struct IdeaCfbContext cfb;
        memcpy(textbuf,str,lenstr);
        mdstr(pw,ideakey);
        ideaCfbInit(&cfb, ideakey);
```

```
        ideaCfbDecrypt(&cfb, textbuf, textbuf, lenstr);
        ideaCfbDestroy(&cfb);
        memcpy(str,textbuf,lenstr);
        burn(textbuf);
        return status;
}
```

PHỤ LỤC:

LƯỢC ĐỒ IDEA

Phần này sẽ trình bầy lược đồ bảo vệ dữ liệu IDEA đã được thiết kế thử nghiệm trong mô hình bảo vệ CSDL. Phần này chủ yếu để phục vụ cho việc theo dõi chương trình được dễ dàng hơn do vậy cơ sở lý thuyết sẽ không được trình bầy ở đây.

1.Những điểm chính

IDEA là phương pháp mã khối sử dụng 128 bit khóa để mã khối dữ liệu 64 bit. IDEA được xây dựng nhằm mục đích kết hợp với nhiều yếu tố khác nhau để tăng độ an toàn và khả năng thực hiện.

* *Độ an toàn:*

- **Độ dài của khối:** khối phải có độ dài đủ để chống lại các phương pháp phân tích thống kê và ngăn việc một số khối nào đó xuất hiện nhiều hơn các khối khác. Mặt khác sự phức tạp của thuật toán tăng theo hàm mũ với độ dài khối. Với khối có độ dài 64 bit là đủ độ an toàn. Bên cạnh đó việc sử dụng chế độ feedback sẽ làm tăng thêm độ an toàn của thuật toán.

- **Độ dài khóa :** Khóa phải đủ dài để có thể chống lại phương pháp vét cạn khóa.

- **Độ phức tạp :** Bản mã phải phụ thuộc một cách phức tạp vào bản rõ và khóa. Mục tiêu đặt ra ở đây là phải làm phức tạp hóa sự phụ thuộc của bộ mặt thống kê của bản mã vào bản rõ. IDEA đạt được điều này nhờ việc sử dụng 3 phép toán sẽ trình bày sau đây.

- **Sự phân bố :** IDEA đã đạt được việc mỗi bit của bản rõ phải có ảnh hưởng đến nhiều bit của bản mã và mỗi bít khóa cũng tác động đến nhiều bit của bản mã. Điều này làm cho cấu trúc của bản rõ sẽ bị phá vỡ trong bản mã.

2.Các phép toán sử dụng trong IDEA

- Phép XOR theo bit. Ký hiệu là $\oplus$

- Phép cộng 2 số nguyên lấy modulo 2^{16} (65536) với đầu vào và đầu ra là 2 số nguyên không dấu 16 bit. Ký hiệu 9.

- Phép nhân 2 số nguyên lấy modulo $2^{16} + 1$ với đầu vào và đầu ra là 2 số nguyên không dấu 16 bit. Qui ước là khối toàn số 0 biểu thị cho 2^{16}. Ký hiệu $\otimes$.

Ba phép toán này thỏa mãn :

- Không có 2 phép toán nào thỏa mãn luật phân phối:

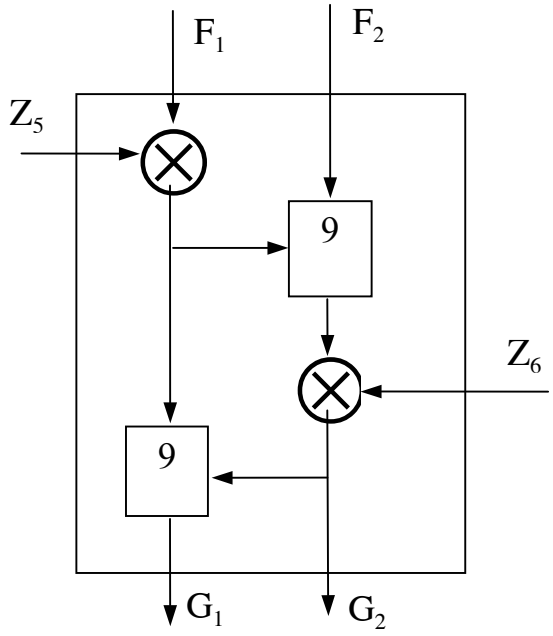
$$a\ 9\ (b \otimes c) \neq (a\ 9\ b) \otimes (a\ 9\ c)$$

- Không có 2 phép toán nào thỏa mãn luật kết hợp:

$$a\ 9\ (b \otimes c) \neq (a\ 9\ b) \otimes c$$

Việc sử dụng kết hợp 3 phép toán này tạo ra một sự biến đổi phức tạp dữ liệu đầu vào làm cho việc mã thám trở nên khó khăn hơn so với việc chỉ sử dụng một phép toán đơn giản.

Trong IDEA sự phân bố được tạo ra dựa trên khối thuật toán có cấu trúc như hình vẽ gọi là cấu trúc MA (Multiplication/Addition).



Hình 1 : Cấu trúc Multiplication/Addition (MA)

Khối này nhận 16 bit từ bản rõ và 16 bit được lấy từ khóa ra theo một qui tắc nào đó (16 bit này được gọi là subkey và qui tắc lấy subkey từ khóa sẽ được trình bày ở sau) để tạo ra 16 bit đầu ra. Một chương trình kiểm tra trên máy tính bằng phương pháp vét cạn xác định rằng mỗi bit ở đầu ra phụ thuộc vào các bit rõ và bit subkey đầu vào. Cấu trúc này được sử dụng lặp lại 8 lần trong thuật toán và tạo nên một sự phân bố có hiệu quả.

IDEA được xây dựng sao cho việc thực hiện nó được dễ dàng cả trên phần cứng và phần mềm. Việc thực hiện trên phần cứng, điển hình là trên vi mạch VLSI, được thiết kế để đạt được tốc độ cao. Việc xây dựng trên phần mềm thì thuận tiện và giá thành thấp.

- Những điểm chủ yếu trong việc xây dựng phần mềm:

+ Sử dụng những khối nhỏ: những phép toán mã thực hiện trên những khối có độ dài 8, 16, 32 bit phù hợp với việc xử lý trên máy tính.

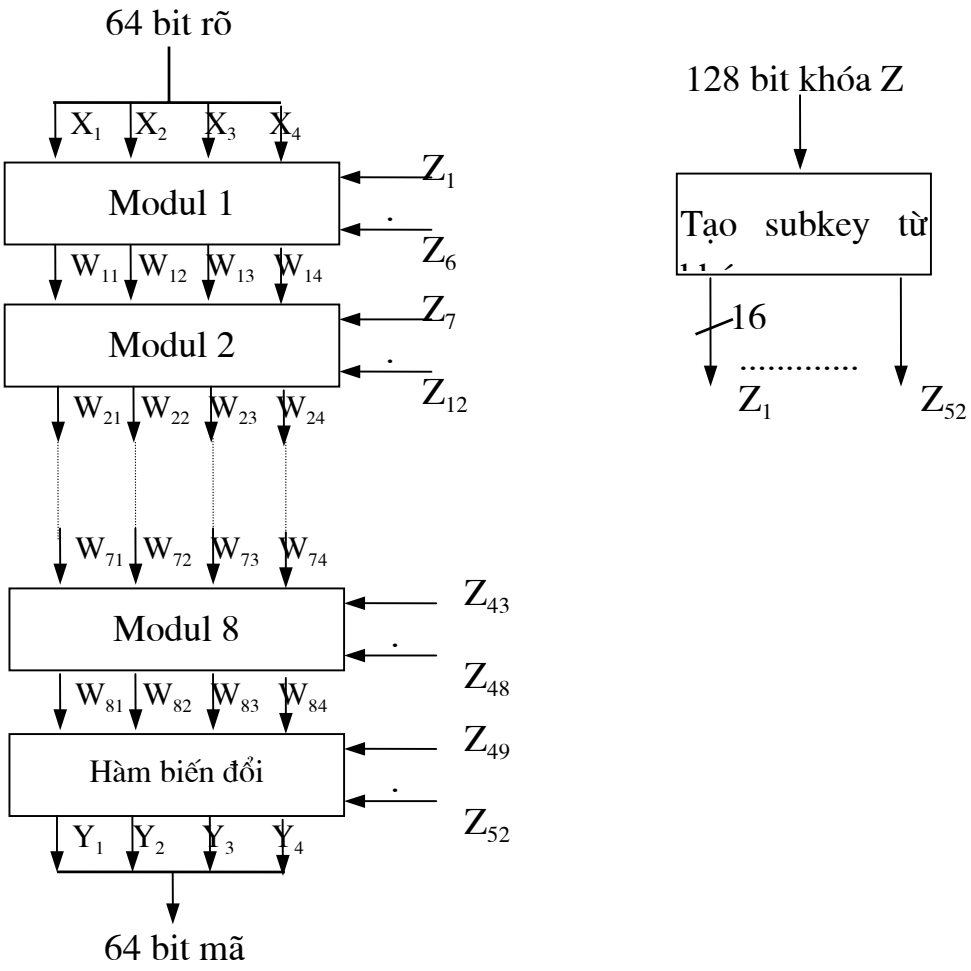
+ Sử dụng thuật toán giản đơn: Phép toán mã dễ dàng trong lập trình như phép cộng, phép dịch chuyển (shift),...Cả 3 phép toán của IDEA đều thỏa mãn những yêu cầu này. Điểm khó khăn nhất là phép toán nhân modulo ($2^{16} + 1$) cũng có thể xây dựng dễ dàng từ những phép toán sẵn có.

- Những điểm chủ yếu trong việc thực hiện trên phần cứng:

+ Sự tương tự trong mã hóa và giải mã: Mã hóa và giải mã chỉ khác nhau trong việc sử dụng khóa và nhờ đó một phương tiện có thể dùng cho cả mã hóa và giải mã.

+ Cấu trúc lặp lại: Phương pháp mã nên có cấu trúc modul lặp lại để các mạch VLSI có thể thực hiện được dễ dàng. IDEA được xây dựng từ hai khối modulo đơn giản và sử dụng lặp lại nhiều lần.

3. Mã hóa và giải mã trong IDEA



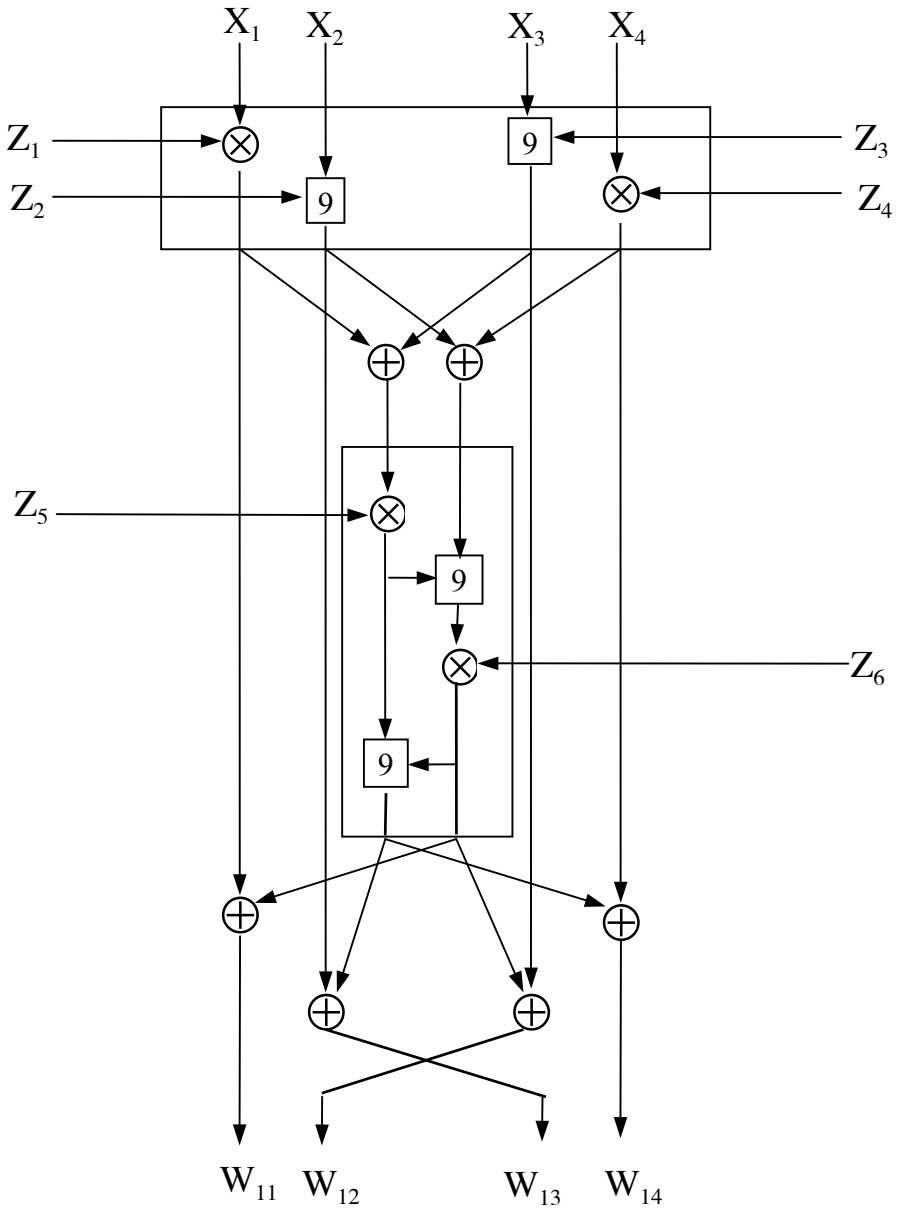
Hình 2 : Cấu trúc của IDEA

a.Mã hóa:

Giống như các sơ đồ mã hóa khác, hàm mã hóa có 2 tham số ở đầu vào là bản rõ cần mã và khóa. Trong trường hợp này là 64 bit rõ và 128 bit khóa.

Từ đầu vào đến đầu ra, các bit rõ lần lượt đi qua 8 modul và một hàm biến đổi cuối cùng. Tám modul này có cấu trúc giống nhau và thực hiện các thao tác như nhau đối với dữ liệu đầu vào. Mỗi modul nhận 4 khối 16 bit rõ ở đầu vào cùng với các subkey và đưa ra 4 khối 16 bit đã được mã hóa. Do đó 64 bit rõ sẽ được chia thành 4 khối nhỏ gọi là các subblock, mỗi subblock là 16

bit. Cùng với các subblock này là 6 khối subkey cũng sẽ được đưa vào từng modul. Như vậy thêm 4 subkey cần thiết cho hàm biến đổi cuối cùng, ta cần tổng cộng 52 khối subkey cho một lần mã.



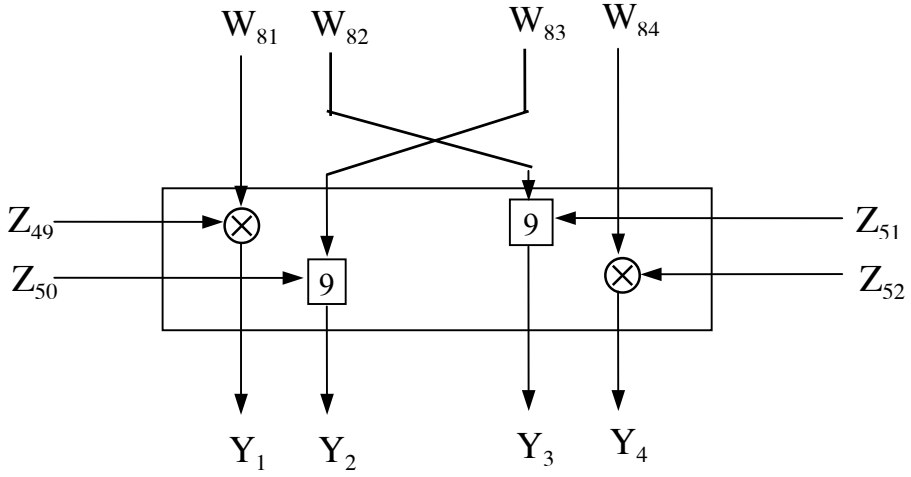
Hình 3 : Cấu trúc một modul

Như đã trình bầy ở trên, các modul có cấu trúc giống nhau và chỉ khác nhau ở dữ liệu đầu vào. Trừ modul đầu tiên nhận 64 bit rõ đưa từ ngoài vào, các modul đứng sau sẽ nhận 4 khối subblock 16 bit đầu ra của modul đứng trước nó làm các bit rõ đầu vào. Trong quá trình đầu tiên các modul kết hợp 4 subblock với 4 subkey bằng các phép toán 9 và ⊗. Bốn khối đầu ra của quá trình này XOR với nhau như trong sơ đồ để tạo ra 2 khối đầu vào cho cấu trúc MA và cấu trúc MA sẽ kết hợp chúng với 2 subkey còn lại để tạo ra 2 khối 16 bit mới.

Cuối cùng, 4 khối được tạo ra từ quá trình đầu tiên sẽ được XOR với 2 khối đầu ra của cấu trúc MA để tạo ra 4 khối đầu ra của modul. Chú ý 2 khối đầu vào X_2 và X_3 đươc hoán đổi cho nhau để

tạo ra 2 khối W_{12} và W_{13} được đưa ra ngoài. Điều này làm tăng sự hòa trộn của các bit được xử lý và tăng khả năng chống lại các phương pháp mã thám.

Hàm biến đổi ở cuối cùng ta cũng có thể coi như là một modul thứ 9. Hàm này có cấu trúc giống như cấu trúc đã thực hiện trong quá trình đầu tiên của một modul chỉ khác là khối thứ 2 và thứ 3 ở đầu vào được đổi chỗ cho nhau trước khi được đưa tới các đơn vị phép toán. Thực ra đây chỉ là việc trả lại thứ tự đã bị đổi sau modul thứ 8. Lý do của việc này là sự giống nhau về cấu trúc của quá trình giải mã quá trình mã hóa.



Hình 4 : Hàm biến đổi của IDEA

*Qui tắc tạo ra subkey:

Như trên đã trình bày, cần thiết phải có 52 khối subkey 16 bit được tạo ra từ 128 bit khóa. Qui tắc tạo như sau:

- 8 subkey đầu tiên, $Z_1...Z_8$, được lấy trực tiếp từ khóa với Z_1 là 16 bit đầu (bit có trọng số cao nhất), Z_2 là 16 bit tiếp theo và cứ tiếp tục như vậy.

- Sau đó khóa được quay trái 25 bit và 8 subkey tiếp theo được tạo ra theo qui tắc trên. Thao tác này được lặp lại cho đến khi có đủ 52 khối subkey.

Qui tắc này là một phương pháp hiệu quả cho việc đa dạng hóa các bit khóa dùng cho các modul. Ta nhận thấy rằng những subkey đầu tiên dùng trong mỗi modul sử dụng những tập hợp bit khác nhau của khóa. Nếu như khóa 128 bit được ký hiệu là Z[1..128] thì subkey đầu tiên của 8 modul sẽ là:

$Z_1 = Z[1..16]$ $Z_{25} = Z[76..91]$

$Z_7 = Z[97..112]$ $Z_{31} = Z[44..59]$

$Z_{13} = Z[90..105]$ $Z_{37} = Z[37..52]$

$Z_{19} = Z[83..98]$ $Z_{43} = Z[30..45]$

Như vậy, 96 bit subkey sử dụng cho mỗi modul, trừ modul thứ nhất và modul thứ 8, là không liên tục. Do đó không có một mối liên hệ dịch chuyển đơn giản nào giữa các subkey của một modul và giữa các modul với nhau. Nguyên nhân có được kết quả này là việc chỉ có 6 khối subkey được sử dụng trong khi có 8 khối subkey được tạo ra trong mỗi lần dịch chuyển khóa.

b.Giải mã

Quá trình giải mã về cơ bản giống quá trình mã hóa. Giải mã nhận bản mã ở đầu vào và cũng đi qua những cấu trúc như ở trên, chỉ khác ở sự lựa chọn các subkey. Các subkey để giải mã U_1, U_2,...U_{52} nhận được từ khóa mã theo qui tắc sau:

- Đối với modul giải mã *i* ta lấy 4 subkey đầu của modul mã hóa thứ (10-*i*), ở đây hàm biến đổi được coi như modul thứ 9. Sau đó lấy nhân đảo modulo (2^{16} + 1) của subkey thứ 1 và thứ 4 để dùng cho subkey giải mã thứ 1 và thứ 4 tương ứng. Đối với các modul từ thứ 2 đến thứ 8, subkey giải mã thứ 2 và thứ 3 là cộng đảo modulo 2^{16} của subkey thứ 3 và thứ 2 tương ứng. Đối với các modul thứ 1 và thứ 9, subkey giải mã thứ 2 và thứ 3 là cộng đảo modulo 2^{16} của subkey thứ 2 và thứ 3 tương ứng.

- Đối với 8 modul đầu tiên, 2 subkey cuối của modul *i* là 2 subkey cuối của modul mã hóa thứ (9 - *i*).

Ở đây nhân đảo Z_j^{-1} của Z_j là phần tử nghịch đảo của Z_j đối với phép toán nhân tức:

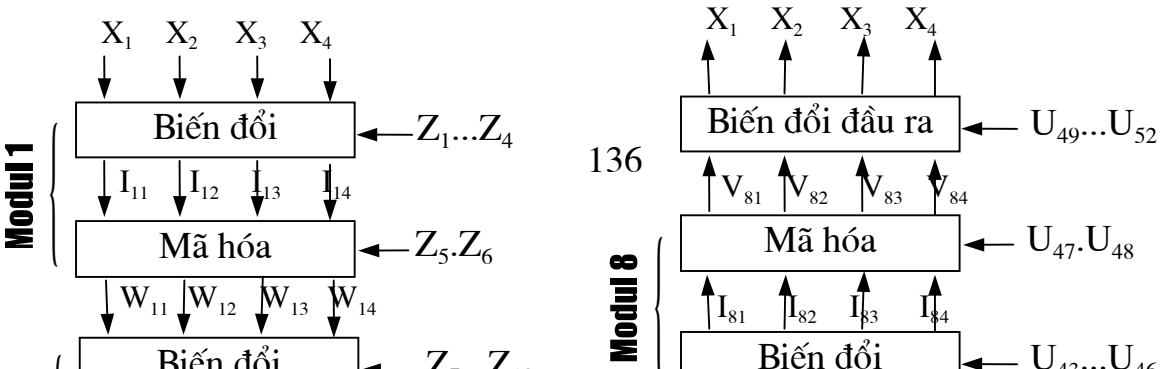
$$Z_j \otimes Z_j^{-1} = 1$$

Vì 2^{16} + 1 là một số nguyên tố nên mỗi số nguyên $Z_j < 2^{16}$ có một số nhân đảo modulo (2^{16} +1) duy nhất.

Với cộng đảo modulo 2^{16} thì:

$$-Z_j \ 9 \ Z_j = 0$$

Hình vẽ sau thể hiện quá trình mã hóa (theo chiều đi xuống bên trái) và quá trình giải mã (chiều đi lên bên phải) của thuật toán IDEA.



Mỗi modul được chia thành 2 khối nhỏ : khối biến đổi và khối mã hóa. Khối biến đổi tương ứng với quá trình đầu tiên trong mỗi modul, còn khối mã hóa tương ứng với các quá trình còn lại. Ở phía cuối của sơ đồ, bên mã hóa ta nhận được các mối quan hệ sau giữa đầu ra và đầu vào của hàm biến đổi:

$$Y_1 = W_{81} \otimes Z_{49} \qquad Y_3 = W_{82}\ 9\ Z_{51}$$

$$Y_2 = W_{83}\ 9\ Z_{50} \qquad Y_4 = W_{84} \otimes Z_{52}$$

Tại khối biến đổi của modul thứ nhất trong quá trình giải mã, đầu ra và đầu vào có mối quan hệ sau:

$$J_{11} = Y_1 \otimes U_1 \qquad J_{13} = Y_3\ 9\ U_3$$

$$J_{12} = Y_2\ 9\ U_2 \qquad J_{14} = Y_4 \otimes U_4$$

Ta có:

$$J_{11} = Y_1 \otimes Z_{49}^{-1} = W_{81} \otimes Z_{49} \otimes Z_{49}^{-1} = W_{81}$$

$J_{12} = Y_2 \,9 - Z_{50} = W_{83} \,9\, Z_{50} 9 - Z_{50} = W_{83}$

$J_{13} = Y_3 \,9 - Z_{51} = W_{82} \,9\, Z_{51} 9 - Z_{51} = W_{82}$

$J_{14} = Y_4 \otimes Z_{50}^{-1} = W_{84} \otimes Z_{50} \otimes Z_{50}^{-1} = W_{84}$

Như vậy, kết quả thu được sau khối biến đổi thứ nhất của quá trình giải mã chính là dữ liệu rõ đưa vào khối mã hóa cuối cùng của quá trình mã hóa chỉ khác là khối dữ liệu thứ 2 và khối dữ liệu thứ 3 đã đổi chỗ cho nhau. Bây giờ ta sẽ xét đến mối quan hệ thu được theo sơ đồ 711:

$W_{81} = I_{81} \,9\, MA_R(I_{81} \,9\, I_{83}, I_{82} 9\, I_{84})$

$W_{82} = I_{83} \,9\, MA_R(I_{81} \,9\, I_{83}, I_{82} 9\, I_{84})$

$W_{83} = I_{82} \,9\, MA_R(I_{81} \,9\, I_{83}, I_{82} 9\, I_{84})$

$W_{84} = I_{84} \,9\, MA_R(I_{81} \,9\, I_{83}, I_{82} 9\, I_{84})$

trong đó $MA_R(X,Y)$ là đầu ra phía bên phải còn $MA_L(X,Y)$ là đầu ra phía bên trái của cấu trúc MA trong hình 79 khi đầu vào là X và Y. Và:

$V_{11} = J_{11} \,9\, MA_R(J_{11} \,9\, J_{13}, J_{12} 9\, J_{14})$

$= W_{81} \,9\, MA_R(W_{81} \,9\, W_{82}, W_{83} 9\, W_{84})$

$= I_{81} \,9\, MA_R(I_{81} \,9\, I_{83}, I_{82} 9\, I_{84}) \,9$

$MA_R[I_{81} 9 MA_R(I_{81} 9 I_{83}, I_{82} 9 I_{84}) 9 I_{83} 9 MA_R(I_{81} 9 I_{83}, I_{82} 9 I_{84}),$

$I_{82} 9 MA_L(I_{81} 9 I_{83}, I_{82} 9 I_{84}) \,9 I_{84} 9 MA_L(I_{81} 9 I_{83}, I_{82} 9\, I_{84})]$

$= I_{81} 9 MA_R(I_{81} 9 I_{83}, I_{82} 9 I_{84}) \,9 MA_R(I_{81} 9 I_{83}, I_{82} 9\, I_{84})$

$= I_{81}$

Tương tự ta có:

$V_{12} = I_{82}$

$V_{13} = I_{83}$

$V_{14} = I_{84}$

Như vậy, kết quả thu được sau khối mã hóa thứ nhất của quá trình giải mã lại là dữ liệu đưa vào khối biến đổi của modul cuối cùng của quá trình mã hóa chỉ khác là khối dữ liệu thứ 2 và khối dữ liệu thứ 3 đã đổi chỗ cho nhau. Cứ như vậy, ta sẽ thu được:

$V_{81} = I_{11}$

$V_{82} = I_{13}$

$V_{83} = I_{12}$

$V_{84} = I_{14}$

Vì hàm biến đổi cuối cùng của quá trình giải mã cũng giống như khối biến đổi trong modul đầu tiên của quá trình mã hóa chỉ khác là có đổi chỗ của khối dữ liệu thứ 2 và khối dữ liệu thứ 3 nên ta có bản rõ thu được sau giải mã giống bản rõ đưa vào mã hóa.