Chương trình KC-01:
Nghiên cứu khoa học phát triển công nghệ thông tin và truyền thông

Đề tài KC-01-01:
Nghiên cứu một số vấn đề bảo mật và an toàn thông tin cho các mạng dùng giao thức liên mạng máy tính IP

Báo cáo kết quả nghiên cứu

PHẦN MỀM BẢO MẬT MẠNG DÙNG GIAO THỨC IP

Quyển 4C: "Phần mềm bảo mật trên môi trường Windows"

HÀ NỘI-2002

Báo cáo kết quả nghiên cứu

PHẦN MỀM BẢO MẬT MẠNG DÙNG GIAO THỨC IP

Quyển 4C: "Phần mềm bảo mật trên môi trường Windows"

Chủ trì nhóm thực hiện:
TS. Nguyễn Nam Hải

MỤC LỤC

MỞ ĐẦU

Đảm bảo sự an toàn của thông tin trên các mạng máy tính là một công việc rất phức tạp. Thông tin trên các mạng máy tính có thể gặp rất nhiều hiểm hoạ từ các hiểm hoạ ngẫu nhiên cho đến những hiểm hoạ cố ý. Tất cả những hiểm hoạ đều dẫn đến mất mát thông tin dưới nhiều góc độ khác nhau. Do vậy bảo vệ thông tin trên các mạng máy tính là một công việc hết sức cần thiết. Công nghệ thông tin càng đi sâu vào cuộc sống thì vấn đề an toàn thông tin càng phải được quan tâm. Tin học hoá toàn bộ các hoạt động của xã hội là một xu thế tất yếu. Trong một xã hội được tin học hoá cao thì vai trò của các hệ thống thông tin máy tính là hết sức to lớn. Bởi vấn đề an toàn thông tin trên các mạng máy tính là một chủ đề tương đối rộng bao hàm nhiều lĩnh vực khác nhau. Cho nên trong điều kiện của nước ta là một nước phụ thuộc hoàn toàn vào công nghệ nhập ngoại thì vấn đề an toàn cũng cần phải được nghiên cứu sao cho phù hợp với hoàn cảnh của chúng ta. Làm thế nào vừa tận dụng được sức mạnh của các hệ thống phần mềm thương mại hiện nay nhưng vẫn kiểm soát được mức độ an toàn của thông tin trên mạng là một trong những vấn đề đáng được quan tâm.

Nội dung nghiên cứu phần này nhằm mục đích nghiên cứu xây dựng giải pháp bảo vệ thông tin trên các mạng máy tính được xây dựng trên nền tảng mô hình mạng Winsock. Mô hình mạng Winsock là một mô hình mạng được phát triển mạnh mẽ sử dụng rộng rãi ngày nay. Do vậy định hướng nghiên cứu vào mô hình này là cần thiết và có ý nghĩa thực tiễn.

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

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

CHƯƠNG 1. 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:

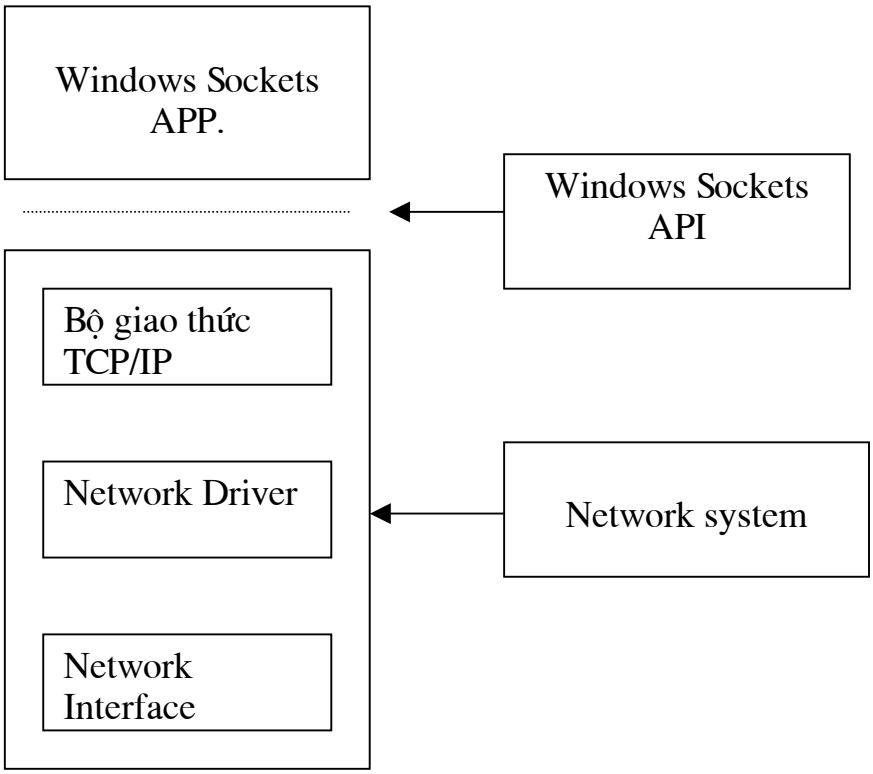
- Winsock là một mô hình được sử dụng rộng rãi hiện nay.
- Winsock là một mô hình mở, cho phép ta can thiệp để đạt được những mục tiêu mong muốn.

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.

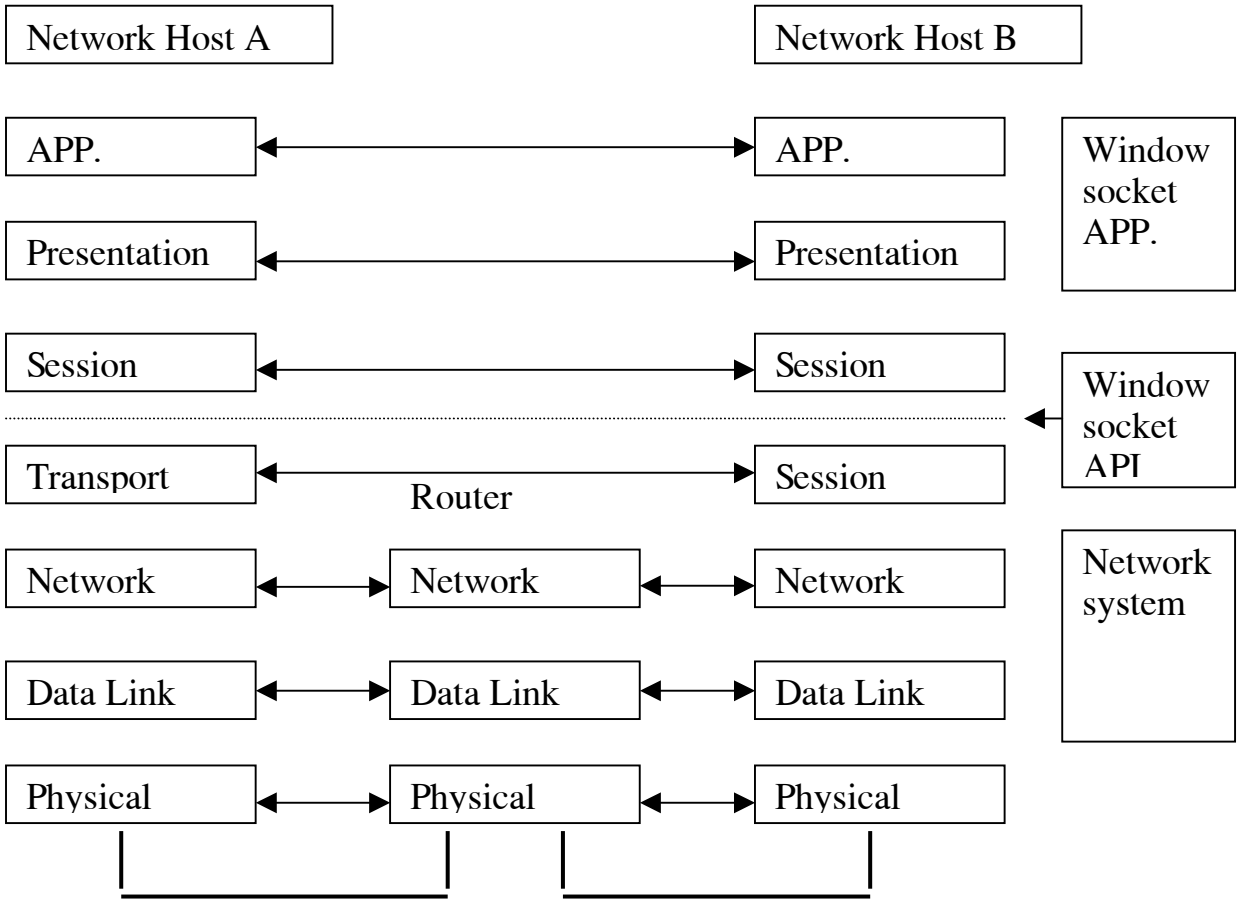
Winsock API (WSA) cung cấp truy nhập tới Network system và các ứng dụng của Winsock sử dụng các dịch vụ của hệ thống để gửi và nhận thông tin.

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.



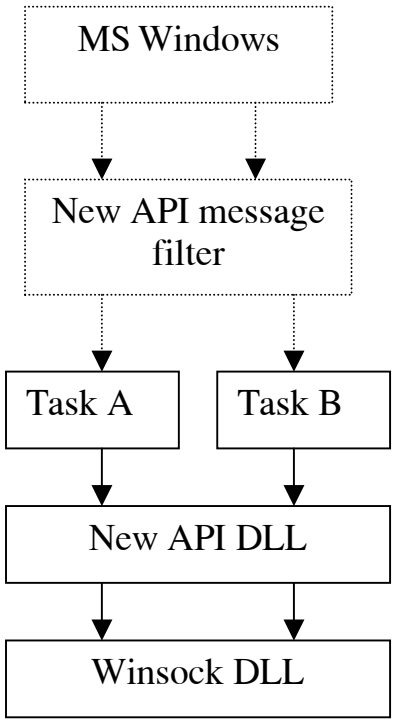
Truyền thông giữa các tầng đồng mức

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 New API DLL

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 giuã 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

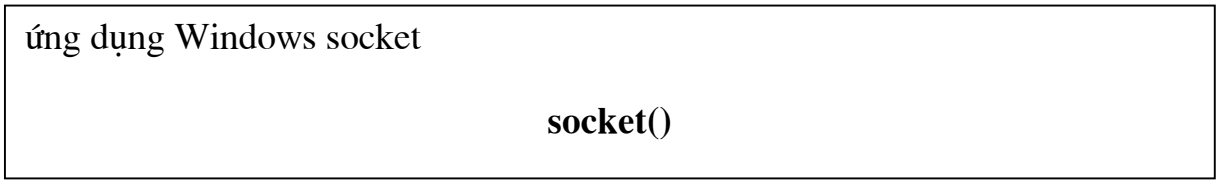
Để các socket tại Client và Server có thể giao tiếp được với nhau thì chúng phải có cùng kiểu. Các ứng dụng Client phải có khả năng xác định và nhận ra socket tại server. Ứng dụng tại server đặt tên socket của nó và thiết lập những đặc tính để nhận diện của nó. Do vậy mà client có thể tham chiếu nó. 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*/
```



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

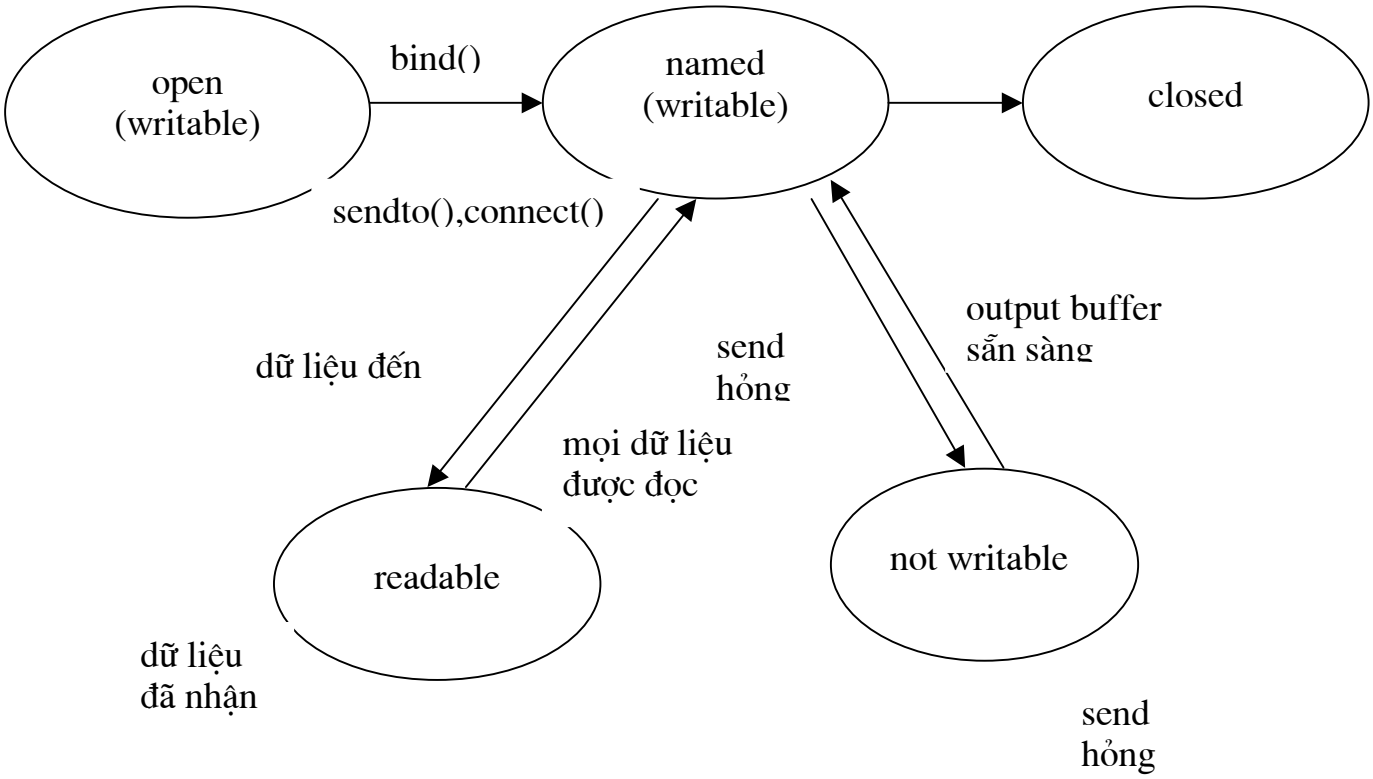
Server cần phải chuẩn bị socket của mình để nhận dữ liệu còn client cần chuẩn bị socket của mình để gửi dữ liệu. Khi việc chuẩn bị xong sẽ tạo ra một liên kết giữa các socket của client và server. Mỗi liên kết là duy nhất trên mạng. Khi liên kết giữa các socket được thiết lập có nghĩa client và server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

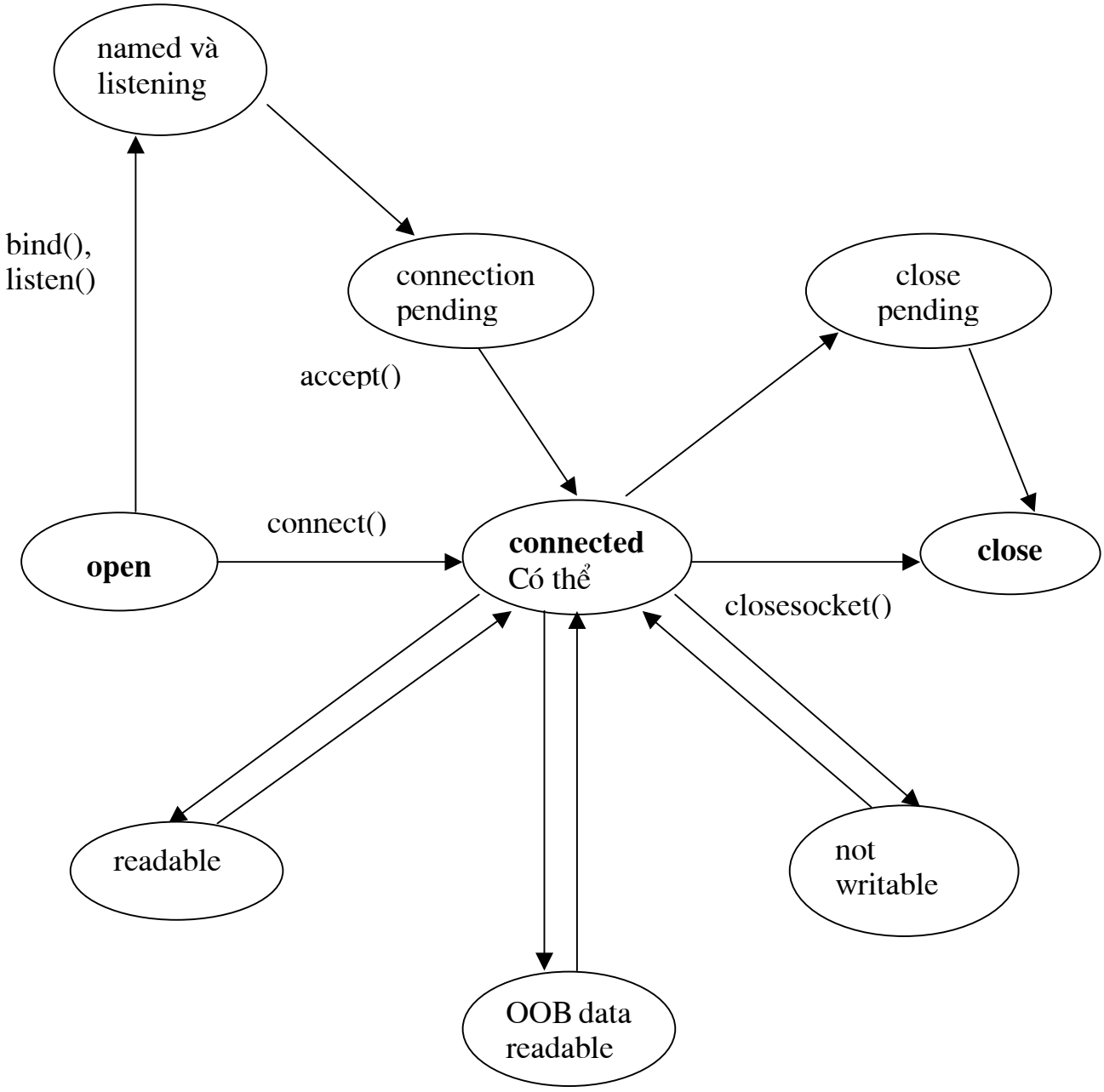


Sơ đồ trạng thái của socket kiểu datagram

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())

CHƯƠNG 2. XÂY DỰNG SOCKET MẬT MÃ

1. Giới thiệu

Chúng tôi phát triển 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 trong FireWall để đảm bảo an toàn và truyền thông tốc độ cao giữa trạm và các máy trạm.

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 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 truy nhập thư điện tử, CSDL tại Server từ các máy cá nhân trên bàn làm việc của mình. Ngày nay, Internet tăng trưởng rất nhanh, nó trở nên quen thuộc đối với các nhân viên khi truy nhập các Server tại công sở của họ từ các trạm từ bên ngoài công sở.

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.

2. 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.

- ***Dễ 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.

3. 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.

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.

4. Thực hiện

4.1. Phương pháp chặn

Có một vài cách để chặn các lệnh của Winsock như sau:

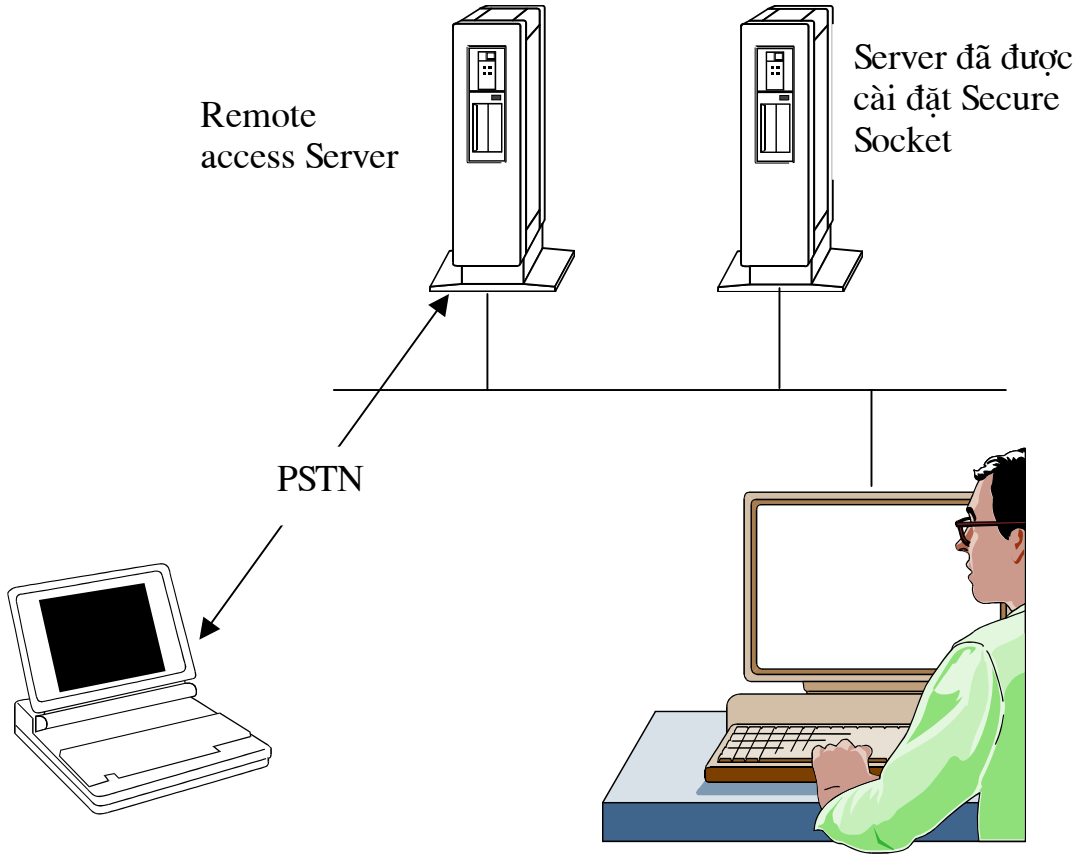
- ***Thay thế các địa chỉ hàm***: Trong một chương trình Windows, phép gọi một hàm được dịch thành một chỉ lệnh nhẩy gián tiếp với địa chỉ của chương trình hàm đích. Do vậy thay thế địa chỉ con trỏ bằng địa chỉ hàm Secure socket tương ứng sau khi tải chương trình ứng dụng. Điều này cho phép Secure socket chặn lời gọi ban đầu.

- ***Thay đổi thông tin liên kết:*** Một file chương trình ứng dụng có thông tin để liên kết tới thư viện Winsock như tên file của nó, số hiệu hàm. Do vậy nếu thông tin liên kết được thay đổi thành liên kết tới thư viện Secure socket thì Secure socket chó thể chiếm điều khiển.

- ***Đổi tên thư viện Winsock***: 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 đầu sẽ rất khó khăn trong việc tính toán thời gian để viết lại con trỏ. Chẳng hạn chạy một ứng dụng ở dạng Debug ta có thể chiếm điều khiển khi ứng dụng bắt đầu và viết lại con trỏ. Nhưng phương pháp này phụ thuộc vào các hệ điều hành. Bằng phương pháp thứ hai thì thông tin liên kết thay đổi theo các Version của các hệ điều hành cũng là một vấn đề.

Chúng tôi sẽ chọn phương pháp cuối cùng do sử dụng 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.

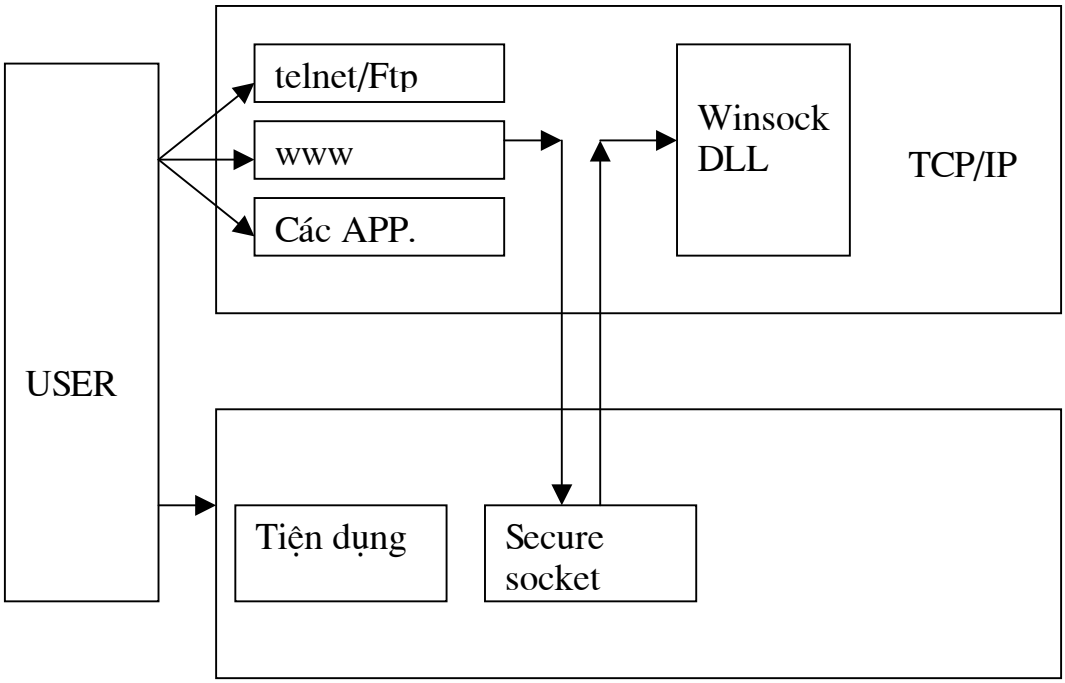


Hình 1. Truy nhập từ xa sử dụng Secure Socket

4.2. 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

4.3. Thao tác kiểu dị bộ

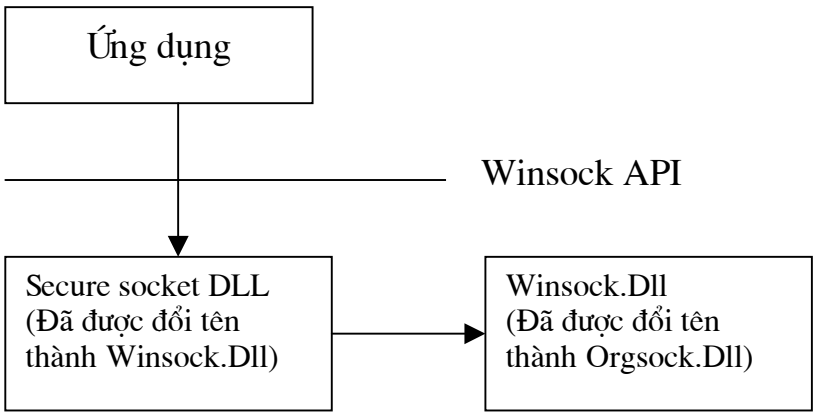
Khi sử dụng các hàm của Winsock, có hai dạng thao tác: Dạng đồng bộ và dạng dị bộ. Các hàm đồng bộ đợi đến khi các phép toán mạng đã yêu cầu được hoàn tất trước khi trả lại lời gọi hàm (lúc đó mới có thể gọi tiếp). Trong khi gọi hàm theo kiểu dị bộ trả lại ngay tức thì mà không quan tâm đến thao tác 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.

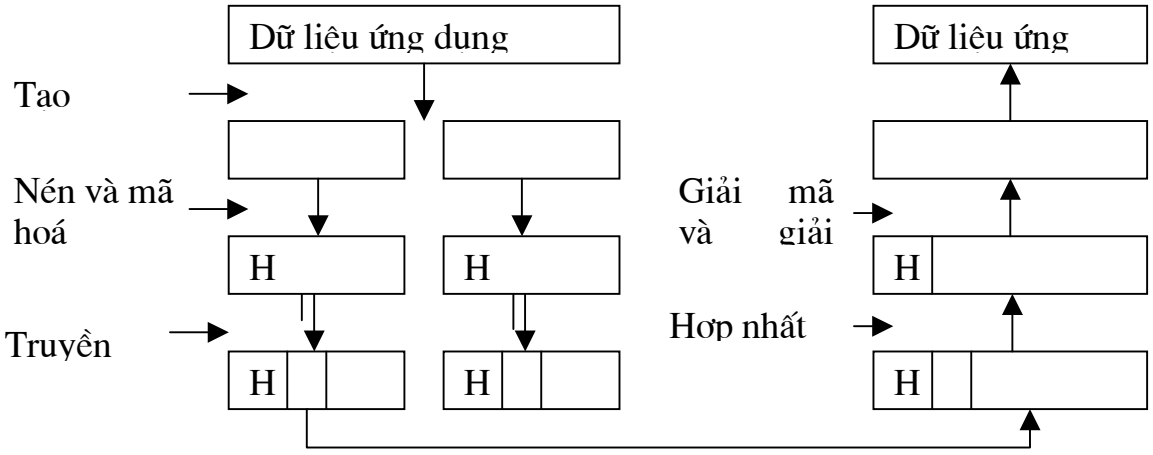
4.4. Thao tác cơ bản

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.

5. 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.

5.1. 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.

5.2. 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 3. LƯỢC ĐỒ MÃ HOÁ IDEA SỬ DỤNG ĐỂ MÃ HOÁ DỮ LIỆU

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 $\boxplus$.

- 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:

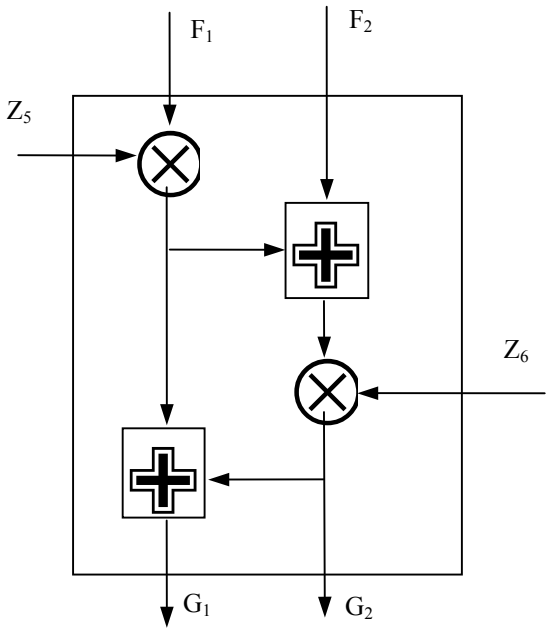
$a \boxplus (b \otimes c) \neq (a \boxplus b) \otimes (a \boxplus c)$

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

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

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.

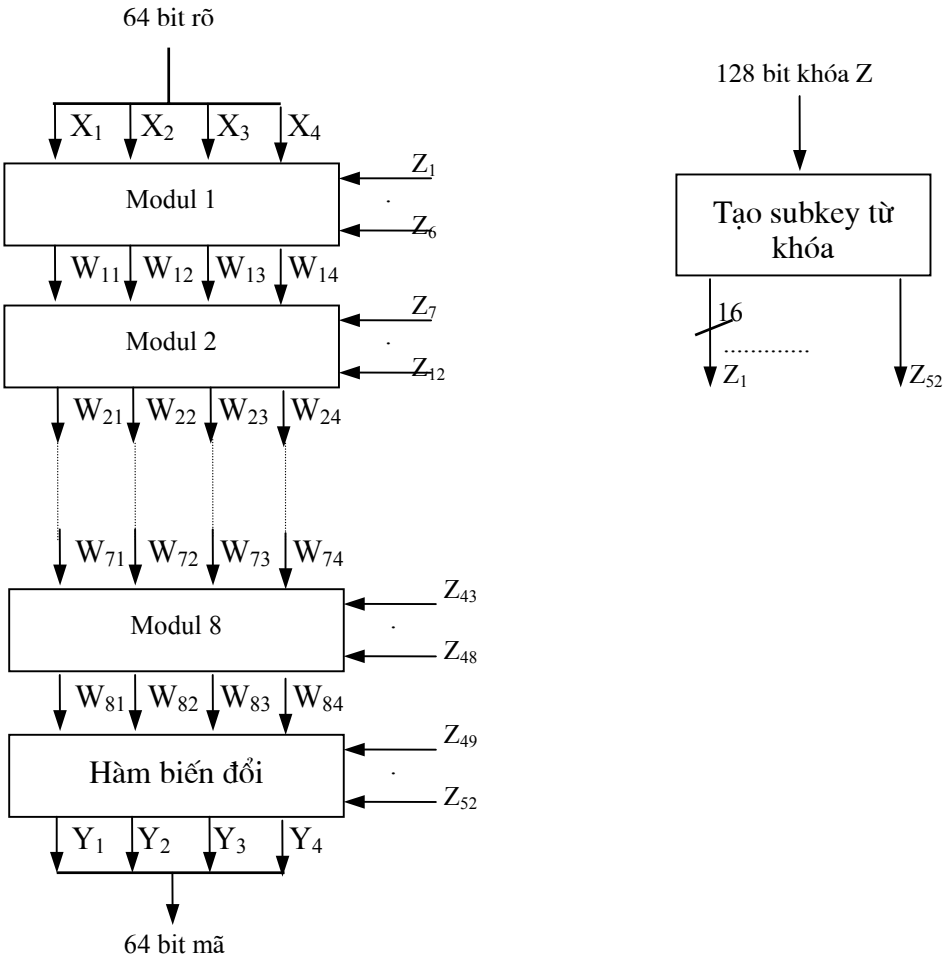
+ Sử dụng thuật toán giản đơn: Phép toán mã dễ dàng trong lập trình như phép cộng, phép dịch chuyển (shift),...Cả 3 phép toán của IDEA đều thỏa mãn những yêu cầu này. Điểm khó khăn nhất là phép toán nhân modulo (2^{16} + 1) cũng có thể xây dựng dễ dàng từ những phép toán sẵn có.

- 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



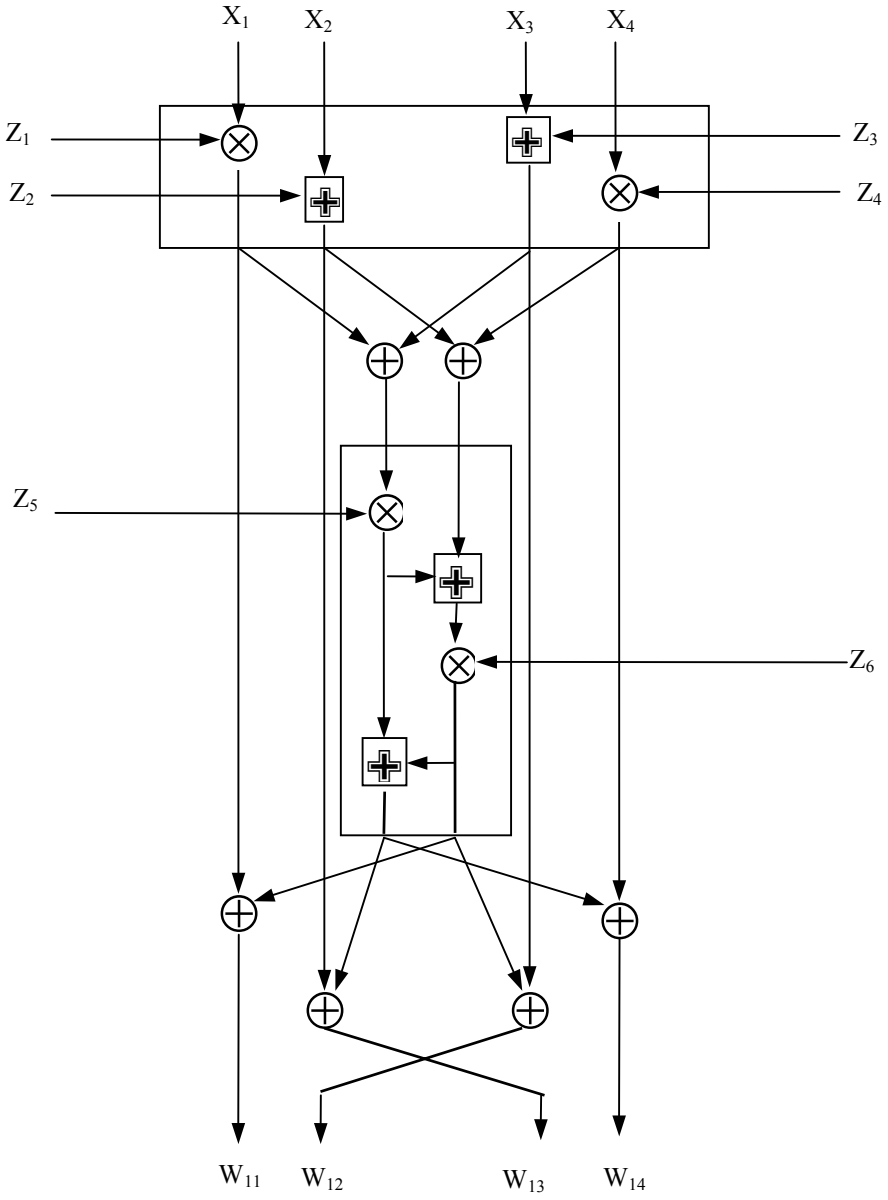
Hình 2 : Cấu trúc của IDEA

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.

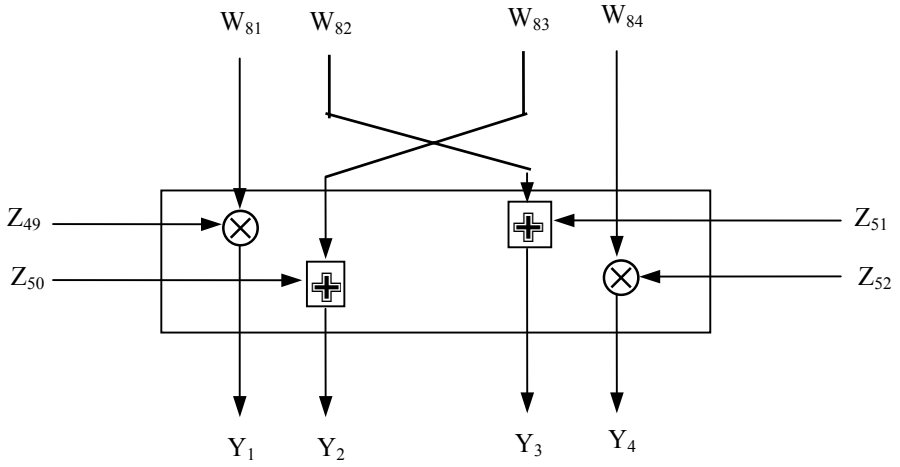


Hình 3 : Cấu trúc một modul

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 ✙ 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 đươc 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:

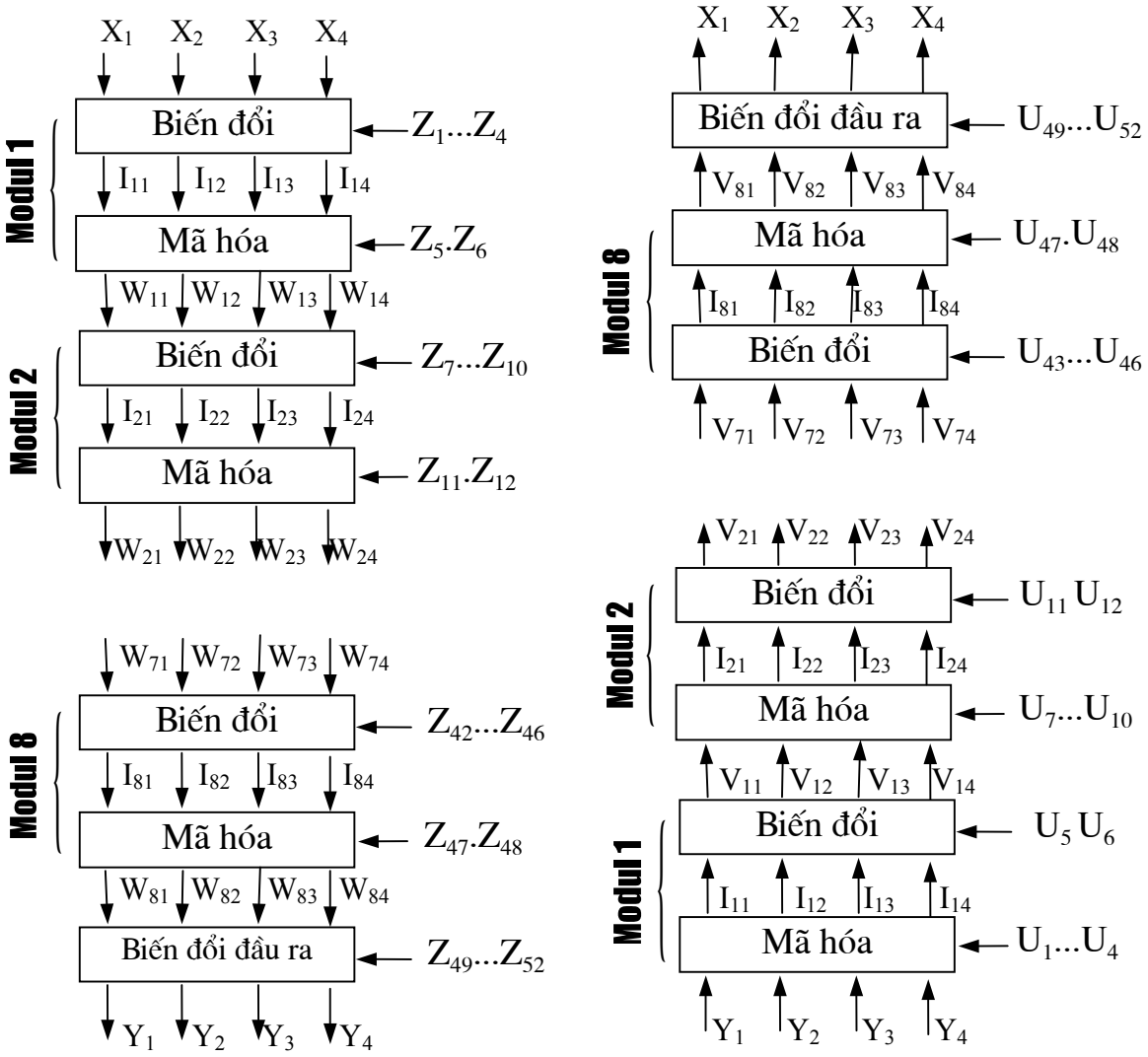
$$Z_j \otimes Z_j^{-1} = 1$$

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:

$$-Z_j \boxplus Z_j = 0$$

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.



Hình 5 : Mã hóa và giải mã trong 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:

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

$$Y_2 = W_{83} \boxplus Z_{50} \quad Y_4 = W_{84} \otimes Z_{52}$$

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

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

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

Ta có:

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

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

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

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

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:

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

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

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

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

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:

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

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

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

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

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

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

$$= I_{81}$$

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.

PHỤ LỤC: PHẦN MỀM SECURESOCKET 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. Phần chương trình thử nghiệm gồm các phần cơ bản sau:

- Các mô đun thuộc socket được thiết kế lại,
- Các mô đun phục vụ cho mã hoá nội dung các gói dữ liệu,
- Các mô đun phục vụ cho việc xác thực nội dung các gói dữ liệu,
- Các mô đun phục vụ cho việc tạo khoá phiên.

Những kỹ thuật mật mã 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 );
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;


        strcpy(trung,"ZZZZZ");
        idea_en_file((unsigned char *)trung,(unsigned char *)buf,len);
        a=GetProcAddress(hModule,"send");
         send1=(int (_stdcall *)(SOCKET ,const char FAR * ,int ,int ))a;
        //abs((char *)buf);
         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;

	//strcpy(trung,"ZZZZZ");

	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,lp
dwStatusFlags);
}

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,lpdwBuffer
Size,lpServiceAsyncInfo);
}
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,dwLocalAddress
	Length,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;
}
```

```
#include <string.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>

#define c 199
#define c1 5
typedef unsigned char BYTE;
typedef unsigned int Word;
typedef BYTE so[c+1];
typedef so mang[c1+1];
```

```
typedef char xau[50];
BYTE io[c1+1]={66,34,18,10,6,4};
mang KEO;
so modulo,somu,mamkhoa;

int behon(BYTE* x,BYTE* y)
{
      int i;

   if (x[c]!=y[c])
      return (x[c]<y[c]);
   i=x[c];
   while (x[i]==y[i] && (i>0)) i--;
   return (x[i]<y[i]);
}

void dodai(BYTE* x)
{
      int i=c-1;

   while (x[i]==0 && i>0) i--;
      x[c]=i;
}

void cong(BYTE* x, BYTE* y)
{
      int i,t=0,j;

   j = x[c]>y[c] ? x[c] : y[c];
      for (i=0;i<=j;i++){
            t=x[i]+y[i]+t;
            if (t>256){
                  x[i]=(BYTE)(t-256);
                  t=1;
            }
            else{
                  x[i]=(BYTE)t;
                  t=0;
            }
      }
   x[c]=j;
}

void tru(BYTE* x, BYTE* y)
{
      int i,t=0;

      for (i=0;i<=x[c];i++){
      t=x[i]-y[i]-t;
            if (t<0){
                  t+=256;
                  x[i]=(BYTE)t;
                  t=1;
            }
            else{
                  x[i]=(BYTE)t;
                  t=0;
```

```
            }
      }
      while (x[i]==0 && i>0) i--;
      x[c]=i;
}

void dich_trai(BYTE* x, BYTE* y, BYTE k)
{
      memset(y,0,c+1);
      for (int i=0;i<=x[c];i++) y[i+k]=x[i];
      y[c]=x[c]+k;
}

void nhan_byte(BYTE *x, BYTE *y, BYTE k)
{
   BYTE i,nho=0;
      Word t;

   memset(y,0,c+1);
   if (k==0) return;
   for (i=0;i<=x[c];i++)
      {
      t=x[i];
      t=t*k+nho;
      y[i]=(BYTE)t;
      nho=t>>8;
      }
   if (nho>0)
      {
      y[i++]=nho;
      }
   y[c]=i-1;
}

void nhan_word(BYTE *x, BYTE *y, Word k)
{
   BYTE i;
      so z,w;

   nhan_byte(x,y,(BYTE)k);
   i = k >> 8;
   if (i==0) return;
   nhan_byte(x,z,i);
   dich_trai(z,w,1);
   cong(y,w);
}

void nhan(BYTE* x, BYTE* y)
{
      Word /*register*/ i,k;
      Word j,r;
      unsigned long t1,t=0;
   BYTE *a,*b;
      so w;

      memset(w,0,c+1);
   a=x; b=y;
```

```
      if (x[c]>y[c])
      {
      a=y; b=x;
      }
   for (i=0;i<=a[c];i++)
      {
      for (k=0;k<=i;k++)
            {
                  t1 = a[k];
                  t += t1 * b[i-k];
            }
      w[i]=(BYTE)t;
            t >>= 8;
      }
      for (i=a[c]+1;i<=b[c];i++)
      {
      for (k=0;k<=a[c];k++)
            {
                  t1 = a[k];
                  t += t1 * b[i-k];
            }
      w[i]=(BYTE)t;
      t >>= 8;
      }
   for (i=b[c]+1;i<=a[c]+b[c];i++)
      {
      j=i-b[c];
      for (k=j;k<=a[c];k++)
            {
                  t1 = a[k];
                  t += t1 * b[i-k];
            }
            w[i]=(BYTE)t;
      t >>= 8;
      }
      w[i++]= t;
   t >>= 8;
   w[i++]= t;
      dodai(w);
   memcpy(x,w,c+1);
}

void binh_fuong(BYTE *x)
{
      Word /*register*/ i,k;
      Word r,s,t1;
      long t=0;
      so w;

      memset(w,0,c+1);
      t=x[0];
   t=t*t;
      w[0]=(BYTE)t;
   t >>= 8;
      for (i=1;i<=x[c];i++){
      s=i >> 1;
            if ((i & 1) == 0){
```

```
                t1 = x[s];
                t += t1*t1;
                s--;
        }
        for (k=0;k<=s;k++){
                r = x[k];
                r = r * x[i-k];
                t += r;
                t += r;
        }
        w[i] = (BYTE) t;
        t >>= 8;
    }
    for (i=x[c]+1; i <= x[c]*2 ;i++)
    {
        s = i >> 1;
        if ((i & 1) == 0){
                t1 = x[s];
                t += t1 * t1;
                s--;
        }
        for (k=i-x[c];k<=s;k++){
                r = x[k];
                r = r * x[i-k];
                t += r;
                t += r;
        }
        w[i] = (BYTE) t;
        t >>= 8;
    }
    w[i++]= (BYTE)t;
    t >>= 8;
    w[i++]= t;
    dodai(w);
    memcpy(x,w,c+1);
}

void du(BYTE *x, BYTE *n)
{
    unsigned long l;
    int i;
    BYTE dn;
    Word a,q,an;
    so v,w;

    dn=n[c];
    an=n[dn];
    an<<=8;
    an^=n[dn-1];
    memset(w,0,c+1);
    if (! behon(x,n) )
        for (i=x[c]-dn;i>=0;i--){
                l=x[dn+i+1];
                l<<=8;
                l^=x[dn+i];
                l<<=8;
                l^=x[dn+i-1];
```

```
                    q=ldiv(l,(unsigned long)an).quot;
                    if (q>0)
                    {
                          dich_trai(n,w,i);
                          nhan_word(w,v,q);
                          if (! behon(x,v)) tru(x,v);
                          else{
                                tru(v,w);
                                tru(x,v);
                          }
                    }
             }
}

void catngan(BYTE *x, BYTE *n)
{
      Word i,j;
      so y,z;

      if (x[c]<=n[c]) return;
      memset(y,0,c+1);
      y[c]=n[c-1];
      memcpy(y,n,y[c]+1);
      memset(z,0,c+1);
      z[c] = x[c]-n[c];
      memcpy(z,x+n[c],z[c]+1);
      j=0;
      while (z[j]==0) z[j++]=255;
      z[j] -= 1;
      for (i=x[c];i>n[c];x[i--]=0);
      x[n[c]]=1;
      x[c]=n[c];
      nhan(y,z);
      tru(x,y);
}

void cat(BYTE *x)
{
      for (int i=0;i<=c1;catngan(x,KEO[i++]));
}

void nhan_mod(BYTE *x, BYTE *y)
{
      nhan(x,y);
      cat(x);
}

void bf_mod(BYTE *x)
{
      binh_fuong(x);
      cat(x);
}

void khoitao(BYTE *n)
{
      Word i,j,k;
      so w;
```

```
	BYTE dn=n[c];
	for (j=0;j<=c1;j++){
		memset(w,0,c+1);
		k=io[j]+dn;
		w[k]=1;
		for (i=0;i<=dn;w[i++]=255);
		w[c]=k;
		memcpy(KEO[j],w,c+1);
		du(w,n);
		tru(KEO[j],w);
		KEO[j][c-1]=dn;
	}
}

void luy_thua(BYTE *x, BYTE *z, BYTE *n)
{
	Word i,j,k;//p,t;
	so y;//w;
	khoitao(n);
	memset(y,0,c+1);
	y[0] = 1;
	for (i=0;i<z[c];i++){
		k=z[i];
		for (j=0;j<8; j++){
			if ((k & 1)==1) nhan_mod(y,x);
			bf_mod(x);
			k >>= 1;
		}
	}
	k = z[z[c]];
	while ( k>0 ){
		if ((k & 1)==1) nhan_mod(y,x);
		bf_mod(x);
		k >>= 1;
	}
	du(y,n);
	memcpy(x,y,c+1);
}
```