CHAPTER 4 DESIGN AND IMPLEMENTATION

4.1 Introduction

A part of the algorithm for the E-PAP System is adopted from FreeSPEKE SDK version 1.02 [48] and SSLeay version 0.9.0b [49]. A client (E-PAP Client) and a server (E-PAP Server) have been designed using Visual C++ version 6 which use this algorithm to implement zero knowledge proof authentication. Table 4-1 is a technical specification for E-PAP System.

Table 4-1: Technical Specification for E-PAP System

Network Requirement	TCP/IP support
Supported Platform	Windows 95, Windows 98, Windows 98 SE, Windows NT4 and Windows 2000.
Supported Authentication Mechanism	
Public Key	Exponential key from Exponential Key Exchange (EKE)
Secret Key	Small shared secret between Client and Server (password)
Supported Industry Standard	
Hash	SHA-1
Interface	PKCS #5 and GSS-API

4.2 Files Distribution

A whole E-PAP system consists of two parties, E-PAP Client and E-PAP Server. Both parties are written in Visual C++ and will dynamically call functions in two

dynamic link library files, `spekebn.dll` and `spekekit.dll`. E-PAP Client has one executable file `client.exe` and E-PAP Server has one executable file `server.exe`, which handle client authentication function. Besides that, a user credential files generator (`generate.exe`) is used to generate user credential. As E-PAP Client is a Microsoft Foundation Class (MFC) application, three shared dynamic link library files `mfco42d.dll`, `mfc42d.dll` and `msvcrtd.dll` are needed for client execution.

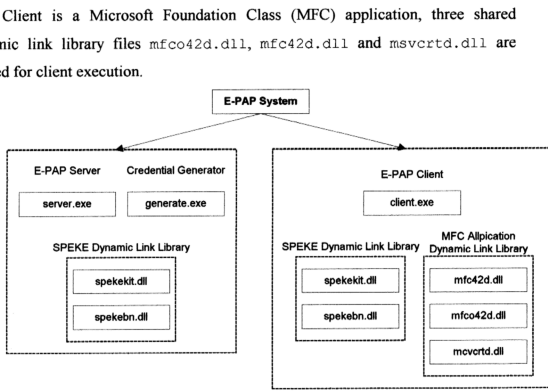

Figure 4-1: E-PAP System files distribution.

4.3 Basic Concept

The client credentials for the API (Application Programming Interface) consist of a username and a password, both represented as null-terminated character strings. The client application establishes a joint security context with the server, using the name and password credentials.

The server credentials consist of a password database. The credential files are stored in a plain file with format of `username.ver`, where `username` is user's logon name and `ver` is the file extension. All the credential files should be located in a folder only reachable by authorised administrator. E-PAP uses the server's knowledge of the user's password verifier to prove the identity of the server to the client. This form

of mutual authentication is a unique feature of zero-knowledge password proofs, which cannot be safely provided by ordinary challenge-hashed-response methods. The server application maintains a password verifier database, in any desired format, which contains a list of names and corresponding password verifiers.

4.4 E-PAP spekebn.dll Library

The `spekebn.dll` file, which is stand for **SPEKE B**ig **N**umber, is modified from SSLeay version 0.9.0b. SSLeay [49] is a Secure Socket Layer (SSL) implementation written by Eric Young (eay@cryptsoft.com). The implementation was written to conform with Netscape SSL. This library is free for commercial and non-commercial use as long as conditions listed in the source-code are adhered to. The current version of this library is available from *http://www.ssleay.org*.

BN is a large integer library. It supports all the normal arithmetic operations. It uses `malloc`[1] extensively and as such has no limits of the size of the numbers being manipulated. The RSA and DH libraries sit on top of this library.

This big number library was written for use in implementing the RSA and DH public key encryption algorithms. This library uses dynamic memory allocation for storing its data structures and so there are no limit on the size of the numbers manipulated by these routines but there is always the requirement to check return codes from functions just in case a memory allocation error has occurred.

The basic object in this library is a `BIGNUM`. It is used to hold a single large integer. This type should be considered opaque and fields should not be modified or accessed directly.

```
typedef struct bignum_st
{
     int top;          //Index of last used d.
```

[1] malloc (memory allocation) is used to dynamically allocate memory at run time.

```
    BN_ULONG *d;     //Pointer to an array of 'BITS^2'
                     //bit chunks.
    int max;         //Size of the d array.
    int neg;
} BIGNUM;
```

The big number is stored in a malloced array of BN_ULONG's. A BN_ULONG can be either 16, 32 or 64 bits in size, depending on the "number of bits" specified in the header file bn.h. d field is this array. max is the size of the d array that has been allocated. top is the last entry being used, so for a value of 4, bn.d[0]=4 and bn.top=1. The neg is 1 if the number is negative. When a BIGNUM is 0, the d field can be NULL and top == 0.

Various routines in this library require the use of 'temporary' BIGNUM variables during their execution. Due to the use of dynamic memory allocation to create BIGNUMs being rather expensive when used in conjunction with repeated subroutine calls, the BN_CTX structure is used. This structure contains BN_CTX BIGNUMs. BN_CTX is the maximum number of temporary BIGNUMs any publicly exported function will use.

```
#define BN_CTX   12
typedef struct bignum_ctx
{
     int tos;              //top of stack
     BIGNUM *bn[BN_CTX];   //The variables
} BN_CTX;
```

For a more detail explanation about big number library, please refer to "SSLeay FAQ" [47].

4.5 E-PAP spekekit.dll Library

FreeSPEKE SDK is an open source code and software developer's toolkit that provides strong password authentication for client/server applications. It uses the

SPEKE™ cryptographic protocol to authenticate a user, and negotiate a mutually authenticated session key to protect the integrity of client/server messages. FreeSPEKE is provided in a highly modular form that enhances portability, and provides freely exportable source code. This open source format facilitates analysis, evaluation and experimental implementation of SPEKE and related methods.

SPEKE is a cryptographic method that protects passwords and related small secrets that are used to authenticate people over a network. It is, in fact, one of the strongest known methods for performing a zero-knowledge proof of knowledge. SPEKE is currently patent-pending, and Integrity Sciences has a very reasonable policy for licensing commercial use of the FreeSPEKE SDK, as well as any other implementations of SPEKE [48].

The `spekekit.dll` file, which stands for **SPEKE Kit**, is adopted from FreeSPEKE SDK version 1.02. There are six main routines for spekekit.dll, and they are summarized in Table 4-2. The detail description of FreeSPEKE API can be found in http://www.freespeke.com.

Table 4-2: FreeSPEKE Routines

C Routine	Function
SpekeNewContext	Creates a new context for a session
SpekeDeleteContext	Deletes a context
SpekeInitServer	Initializes a SPEKE server.
SpekeInitClient	Initializes a SPEKE client.
SpekeReleaseBuffer	Releases a token assigned by a FreeSPEKE function.
SpekeInitBuffer	Initializes a token to be assigned by FreeSPEKE.

The **SpekeNewContext** function creates a new uninitialized context, and is the first step in establishing a session. A call to this function is typically followed by several calls to Init function like **SpekeInitClient** or **SpekeInitServer**, which establish and negotiate the shared information that makes up the security context. These functions

are used to perform the SPEKE handshake, which provide mutual authentication between client and server, and establishes an authenticated session key.

Each Init function can return a token, which the calling client or server application must send to its peer, and the peer in turn passes the token to its Init function. After Init decodes a token, it may tell the application to send another token to continue the process of context establishment. Eventually, the Init calls on both sides tell each application when the handshake is complete, and whether or not it was successful.

At the end of a session, client and server applications each call **SpekeDeleteContext** to delete the security context. After performing mutual authentication, for subsequent communication, FreeSPEKE provides additional functions to:

- create a message authentication code (MAC), which provides integrity protection and origin authentication for the message, and
- verify a MAC for a message created by the peer application.

FreeSPEKE treats messages as arbitrary byte arrays. The transmitting application calls the **SpekeCreateMAC** routine to create a MAC token for a message, using the security context, and typically sends the MAC token along with the message to the receiving application. The receiver passes the message and MAC tokens to the **SpekeVerifyMac** function that validates the data.

The message authentication function used is the HMAC-SHA1 construction [53]. The construction is similar to that used in PKCS #5 [54], except that the negotiated SPEKE key is used in the key for the hash to prevent brute-force attacks. The FreeSPEKE API is much simpler than GSS-API [65], but provides its functionality in a similar manner. The API provides an abstract way to authenticate and negotiate a security context for a session and to authenticate messages based on the security context.

4.6 E-PAP Server

E-PAP Server (`server.exe`) has a default port number 51688 that has been chosen arbitrarily by author. RFC 1700 [59] contains a list of port number assignments from the Internet Assigned Numbers Authority (IANA). The port numbers are divided into three ranges:

- The *well-known ports*: 0 through 1023. These port numbers are controlled and assigned by the IANA. For example, port 80 is assigned for a Web server, port 21 is assigned for a FTP server.
- The *registered ports*: 1024 through 49151. These are not controlled by the IANA, but the IANA registers and lists the uses of these ports as a convenience to the community. For example, ports 6000 through 6063 are assigned for an X Window server.
- The *dynamic* or *private* ports: 49152 through 65535. The IANA says nothing about these ports. As a result, E-PAP Server has arbitrary chosen port number 51688.

An E-PAP Client which wants to establish a connection with a E-PAP Server have to connect through the server host name and this default port number. The server host name is used to resolve IP address for the server. The two values that identify server and client - an IP address and a port number, are often called a *socket* [58]. The *socket pair* for a connection is the two endpoints of the connection: the local IP address, local port, foreign IP address and foreign port. A socket pair uniquely identifies every connection on a network.

E-PAP Server will handle all client authentication by opening a listening socket, accept connections, read user logon name, lookup user's verifier, perform E-PAP handshake (Refer to Figure 4-9) to authenticate and initiate a session and process authenticated Remote Procedure Call (RPC) (Refer to Figure 4-10). If a user wants to change his/her password, he/she must be authenticated first, then create a new credential with the new password and sends the credential to E-PAP Server (Refer to Figure 4-5). The main routines for E-PAP Server have been described in Section 4.6.2 to 4.6.7.

Two simple audit log files (E-PAP System Invalid Password Log and E-PAP System Invalid User Log) have been added to increase E-PAP Server performance and security. The log files are in plain text format with filename `logxpw.txt` and `logxuser.txt` respectively. The former is used to log any invalid password to prevent online brute-force attack and the latter is used to prevent any possible timing-based name guessing attack. Both log files will capture

- logon date,
- logon time,
- client's IP address,
- port number and
- attempt logon name.

There are no maximum file size limit for current the version and the log files will be cleared manually when required. Besides that, E-PAP Server is able to reject any request from a particular client IP address. E-PAP Server will block any IP addresses that have been added in file `blockIP.txt`.

4.6.1 Requirements

The system requirements that are needed to install the E-PAP Server are shown in Table 4-3. The main routines for E-PAP Server have been described in Section 4.6.2 to 4.6.7.

Table 4-3: Installation Requirement for E-PAP Server

Element	Requirement
Processor Speed	PC with a 486/66 MHz or higher processor (Intel Pentium 90 processor and above recommended)
RAM	32 MB and above
Operating system	Windows NT 4 (SP3) and above

Network	TCP/IP
Privileges	Administrator or Root
Hard-disk space	Approximately 1 MB
Code	Visual C ++ 6.0 and above
Display	VGA or higher resolution monitor

4.6.2 checkIP

```
int checkIP (
      FILE* fin,
      char* buf
)
```

Purpose:

Reject any request from particular client IP address that has been added in `blockIP.txt` to prevent attacks.

Parameters:

fin	Open the file *fin (`blockIP.txt`)
buf	Temporary buffer that read and compare client IP address

Function value:

None.

4.6.3 processChangePw

```
int processChangePw (
      char* buf
)
```

Purpose:

Process all password-changing requests from client.

Parameters:

buf	Temporary buffer that store user logon name and credential.

Function value:

None.

4.6.4 doServerHandshake

```
int doServerHandshake(
      SOCKET newSock,
      SpekeContextHandle handle,
      SpekeToken * pVerifierToken
)
```

Purpose:

To initiate a secure context for a session with a client.

Parameters:

newSock	New socket description.
handle	SpekeContextHandle Context handle for the session
pVerifierToken	SpekeToken Token containing the user's password verifier

Function value:

SPEKE_COMPLETE	Successful completion
SPEKE_CONTINUE_NEEDED	Indicates that a token from the client is

	required to complete the context, and that SpekeInitServer must be called again with that token.
SPEKE_E_NO_CRED	Consistency checks performed on the password failed.

4.6.5 processAuthenticatedRPC

```
int processAuthenticatedRPC(
      SOCKET newSock,
      SpekeContextHandle handle
)
```

Purpose:

Get a message, and its MAC code, verify it and send a new MAC for the modified reply.

Parameters:

newSock	New socket description
handle	SpekeContextHandle Context handle for the session

Function values:

None

4.6.6 processNewConnection

```
int processNewConnection(
      SOCKET newSock
)
```

Purpose:

Read user name, lookup user's verifier, perform SPEKE handshake to authenticate and initiate session, and processAuthenticatedRPC() test messages.

Function values:

SPEKE_E_NO_CRED	Consistency checks performed on the password failed.

4.6.7 runServer

```
int runServer(
    short port
)
```

Purpose:

Open a listening socket, accept connections, and processNewConnection() for each.

Parameters:

port	Port number

Function values:

SPEKE_E_NO_CRED	Consistency checks performed on the password failed.
SPEKE_SUCCESS	Successful completion

4.7 E-PAP Client

E-PAP Client (`client.exe`) is used to authenticate itself to E-PAP Server (Figure 4-4) and to change user password (Figure 4-3). The main routines for E-PAP Client have been described in Section 4.7.2 to 4.7.7.

Some client program like telnet is in DOS command prompt. Our aim users' computer literacy may only have naive or novice skill, they may found the command-line programs supported by DOS difficult to use. As a result, E-PAP

Client program has a user-friendly Graphical User Interface (GUI). GUI is required for best visual effect. The system should provide easy navigation, meaningful messages to help users use the system with more confidence.

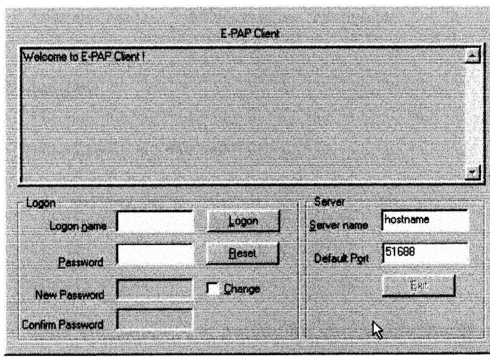

Figure 4-2: E-PAP Client

4.7.1 Requirements

The system requirements that are needed to install the E-PAP Client are shown in Table 4-4.

Table 4-4: Installation Requirement for E-PAP Client

Element	Requirement
Processor Speed	PC with a 486/66 MHz or higher processor (Intel Pentium 90 processor and above recommended)
RAM	16 MB and above
Operating system	Windows 95 and above

Network	TCP/IP
Privileges	Basic access user
Hard-disk space	Approximately 8 MB
Display	VGA or higher resolution monitor
Pointing device	Microsoft mouse or compatible

4.7.2 changePw

```
int CClientDlg::changePw
(
      CString name,
      CString m_NewPassword
)
```

Purpose:

Use to change user password. Only password that allow by **detectBadPw** routine (Section 4.7.3) can be chosen.

Parameter:

name	User logon name.
m_NewPassword	User new password

Function values:

None.

4.7.3 detectBadPw

```
int CClientDlg::detectBadPw
(
      CString name,
      CString m_NewPassword
)
```

Purpose:

Use to detect bad-chosen password, includes:

- password consist of numeric only,
- password contains logon name,
- password contains ≥ 4 same characters
- password contains > 4 following sequent characters and
- password contains a word from English dictionary.

Parameter:

name	User logon name.
m_NewPassword	User new password

Function values:

None.

4.7.4 doClientHandshake

```
int CClientDlg::doClientHandshake
(
     SOCKET sock,
     SpekeContextHandle * handle
)
```

Purpose:

Use to initiate secure context for session with server by swapping tokens until SPEKE says we are done, or it fails.

Parameter:

sock	Socket description
handle	SpekeContextHandle Context handle for the session

Function values:

SPEKE_SUCCESS	Successful completion
SPEKE_CONTINUE_NEEDED	Indicates that a token from the client is required to complete the context, and that SpekeInitServer must be called again with that token.

4.7.5 authenticatedRPC

```
int CClientDlg::authenticatedRPC
(
      SpekeContextHandle handle,
      SOCKET sock,
      CONST char * msg
)
```

Purpose:

Send a message and it is corresponding MAC to the server and receive a MAC of the modified reply.

Parameters:

handle	SpekeContextHandle Context handle for the session
sock	Socket description
msg	Message to be sent to server

Function values:

None.

4.7.6 doClient

```
int CClientDlg::doClient
(
      SOCKET sock
)
```

Purpose:

Initiate secure session and send a series of test messages.

Parameters:

sock	Socket description

Functions values:

None.

4.7.7 runClient

```
int CClientDlg::runClient
(
      char * cHostName,
      char * sHostName,
      int port
)
```

Purpose:

To make TCP connection to server, and call doClient function.

Parameters:

cHostName	Host name for client
sHostName	Host name for server
port	Port number

Function values:

None.

4.8 Procedure Sequence Flow Chart

The E-PAP system is divided into two major modules: change password module and authentication module, which is shown in Figure 4-3 and Figure 4-4. For detail procedure, refer to Figure 4-5 and Figure 4-8.

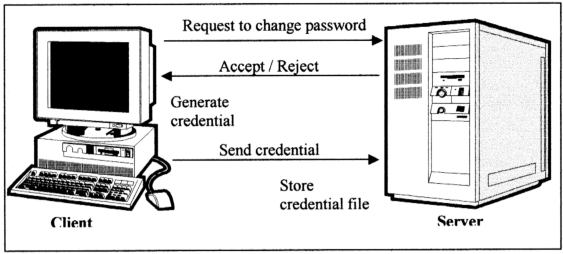

Figure 4-3: Change password module

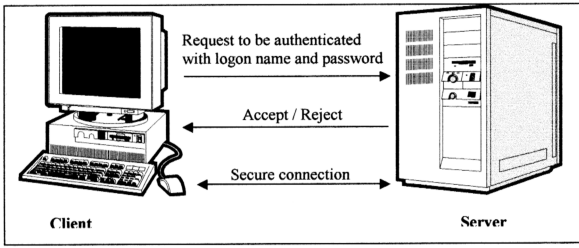

Figure 4-4: Authentication module.

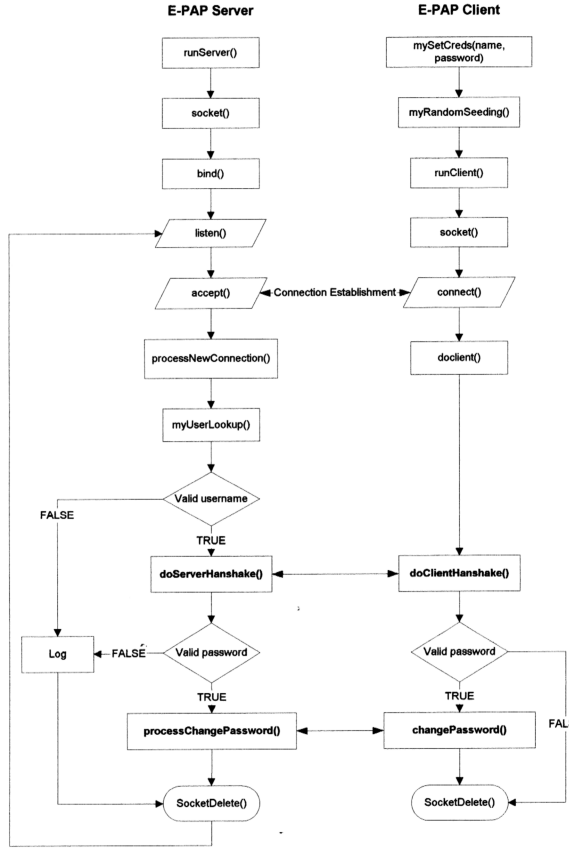

Figure 4-5: Change password procedure flow between E-PAP Server and Client

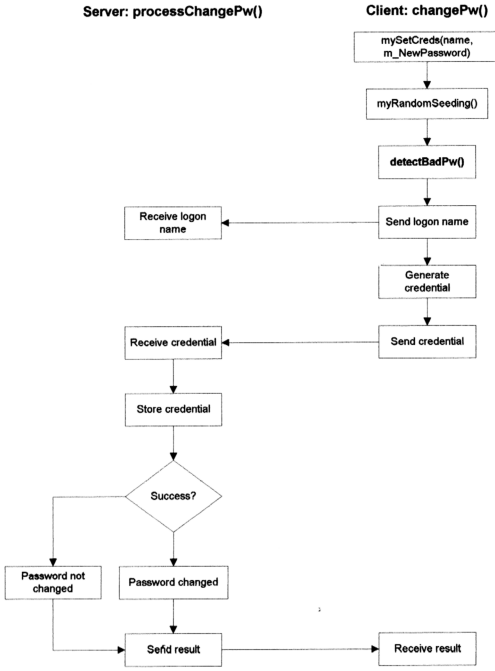

Figure 4-6: Detailed processChangePassword() and changePassword().

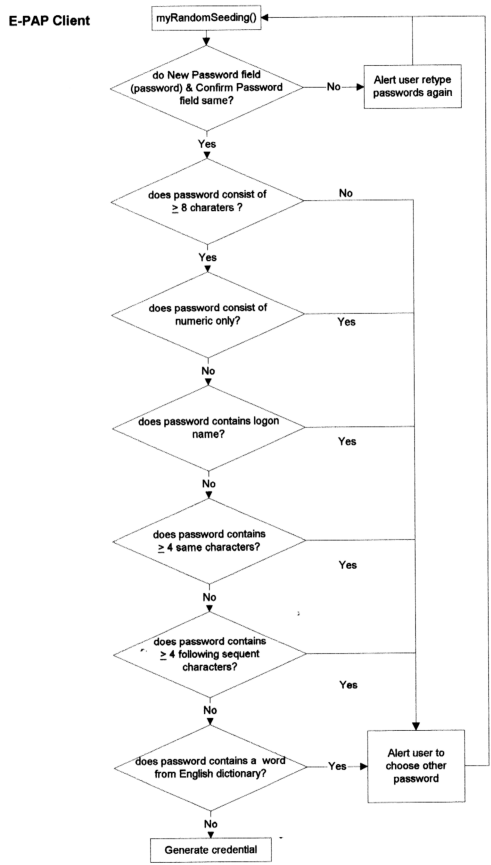

Figure 4-7: Detailed detectBadPw() routine.

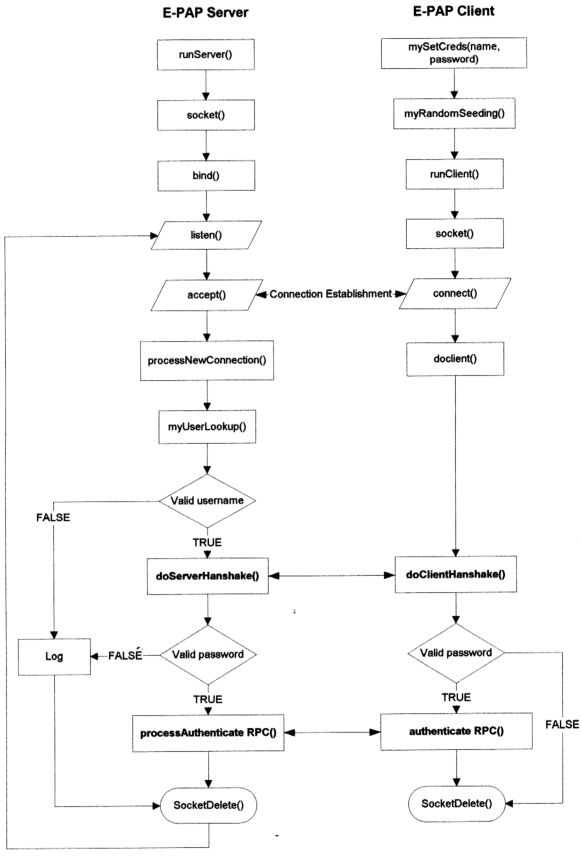

Figure 4-8: Authentication procedure flow between E-PAP Server and Client

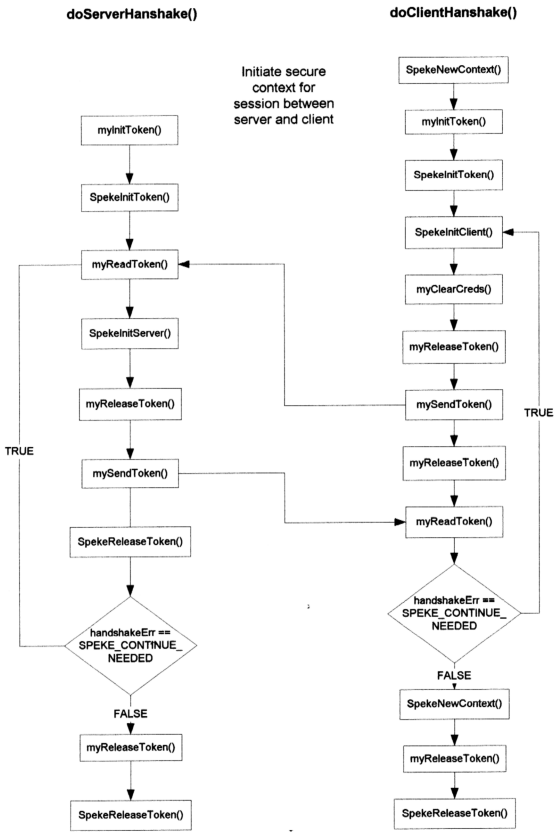

Figure 4-9: Detailed handshake flow between Server and Client.

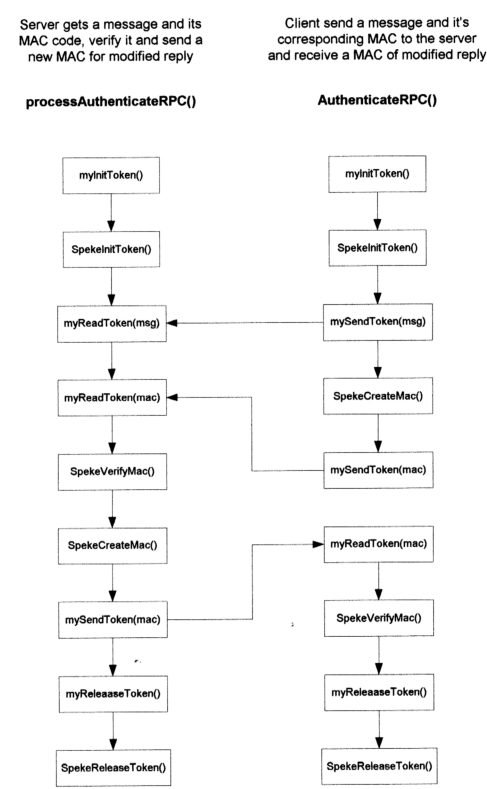

Figure 4-10: Detail Authenticated RPC flow between Server and Client

4.9 E-PAP Server Implementation

E-PAP Server involves

- two executable files (`server.exe and generate.exe`),
- two dynamic link libraries (`spekekit.dll` and `spekebn.dll`) and
- two audit log files (`logxuser.txt` and `logxpw.txt`)
- one blocking IP file (`blockIP.txt`)

These files stored in a folder / directory that should be accessible only by the administrator. To start E-PAP Server, an administrator has to type `server` in dos command, dos prompt or click its shortcut on start menu, and administrator will be prompted to provide a port number. Default port number for E-PAP Server is 51688. To stop the server, the administrator has to press Ctrl + C.

The user credential files generator (`generate.exe`) is used to generate user credential. To register a new user, the administrator have to type `generate` in dos command prompt or click its shortcut on the start menu. After that, the administrator will be prompted to provide a user logon name. That logon name will be registered in `user.txt` file and a credential file with a password with eight alphanumeric characters that is randomly generated by system. The credential file will have filename `username.ver` where `username` is the logon name and `ver` is the file extension. That user will be informed of his/her logon name and password by the administrator, and he/she may change his/her password by using an E-PAP Client. Any invalid password and invalid user name from user / attacker will be captured in `logxpw.txt` and `logxuser.txt` respectively. These log files will be automatically generated and stored in the folder / directory same as E-PAP Server executable file and should only accessible by administrator.

4.10 E-PAP Client Implementation

E-PAP Client is used to replace original Client For Microsoft Network Logon (Figure 4-12) for Windows 95/98/98 Second Edition (SE), Microsoft Family Logon (Figure 4-13) for Windows 98/98 SE, NT LAN Manager (NTLM) for Windows NT 4, Kerberos version 5 for UNIX and Windows 2000. However, E-PAP Client only supports platforms from Microsoft Windows family.

Notes that Windows 9x can be restricted from someone that does not have permission (does not have user name and password) to log on by just clicking Cancel button in Log on dialog box or simply press Escape key (Figure 4-11). This is achieved by setting below key in Registry:

```
[HKEY_LOCAL_MACHINE\Network\Logon]
"MustBeValidated"=dword:00000001
```

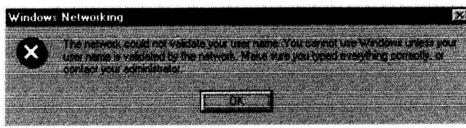

Figure 4-11: Windows restrict logon access.

Windows NT and 2000 platform are restricted to compulsory logon and user is not permitted to access a computer before successful logon.

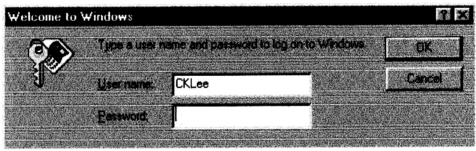

Figure 4-12: Client for Microsoft Logon

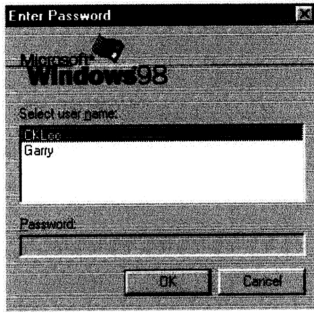

Figure 4-13: Microsoft Family Logon

All files required by E-PAP Client, include

- one executable file (`client.exe`),
- two dynamic link library files, `spekebn.dll` and `spekekit.dll`,
- one dictionary file `English.txt`, which prevent user to choose a password that contain a word from dictionary. This could protect from on-line dictionary and off-line dictionary attack.
- three shared dynamic link library files: `mfco42d.dll`, `mfc42d.dll` and `msvcrtd.dll`, which are needed for MFC application,

have been compressed into a installation file. After installation and restarting system, E-PAP Client will be executed before user log on to a computer. All the testing and the result for E-PAP Client and E-PAP Server had been carried out and recorded in Chapter 5.