An Approach for Intranet Data Security

Prof. Samir Kumar Bandyopadhyay
Dept. of Computer Sc. & Engg, University of Calcutta
92 A.P.C. Road, Kolkata – 700009, India

Suman Chakraborty
B.P. Poddar Institute of Management and Technology
137, V.I.P. Road, Kolkata – 700052, India

ABSTRACT

In this paper, combined the encrypting and compressing processes to consider the choices of which types of bits are most effective in the selective encryption sense when they are changed. So, instead of encrypting the whole file bit by bit, changed only these highly sensitive bits. Moreover, by combining the compression and encryption tasks and reducing the total encryption work required, it will helps to achieve a savings in system complexity. Selective encryption is the technique of encrypting some parts of a compressed data file while leaving others unencrypted. Selective encryption is not a new idea. It has been proposed in several applications, especially in the commercial multimedia industry. Hover, selective encryption of lossless compressed text files has not been explored, and that is the focus of this paper.

Keywords

Codeword, encryption, compression, alphabet, security, frequency, Huffman coding

1. INTRODUCTION

Security has long been seen as a major sticking point in the adoption of Internet technology in the enterprise. As networks have grown and connected to the Internet, the spectre of the hacker has haunted managers responsible for both delivering information within the enterprise and to its partners, and protecting it from unauthorised outsiders.

In fact, the security capabilities of the latest Internet and intranet technologies enable companies to control the availability of information and the authenticity of that information better than ever before. The increasing sophistication of both server and client software means that this unprecedented level of security can be provided without requiring users to undergo complex and bureaucratic procedures to gain legitimate access to sites.

Firewalls also offer some protection to users venturing out from the network to the Internet, acting as proxies to fetch web pages so that the name and IP number of machines on the network are not revealed to web sites that they visit-preventing hackers from learning details of the structure of the network.

While the basic firewall remains a fundamental of Internet and intranet security, increasing levels of sophistication are required by many users as access to the corporate intranet needs to be widened beyond those physically present on the same network. Allowing users dial-up access behind the firewall violates basic security principles; restricting them to the same access offered to the rest of the Internet in front of the firewall denies them valuable services[1,2].

A potential weakness of VPN solutions is their relative inflexibility. VPNs work well for creating fixed tunnels from one known point to another, but they are less well suited to situations where access needs to be given on-the-fly to groups of people not necessarily known at the outset, or who need to gain access from a variety of locations. VPN technology at present works best for encrypting traffic between two known points that are accepted as valid destinations for traffic: once a link has been established, the technology is used to encrypt the information which is sent, not for establishing the validity of the destination to which it is being sent.

As more flexible VPN access is required, the prime issue becomes that of authenticating potential visitors to the site and the credentials that they present. Are they who they say they are, or an impostor? With this capability it is possible to open up the system to provide access to a wider range of partners, customers or suppliers.

The use of public-key based security systems requires considerable care in system design and management. The security of the entire system is ultimately guaranteed by the security of the key used for signing certificates at the top (commonly called the root) of the public key infrastructure. Here specialized hardware can play a useful role.

Normally, all keys that are accessed by the server are held at some point in the main memory of the server, where they are potentially vulnerable to attack (for example, in a server core dump). A higher degree of protection is desirable for the most valuable keys.

A specialized hardware cryptographic module for storing and protecting the signing keys provides an answer. The keys are stored in a strongly encrypted format. When loaded for signing, the keys are decrypted and loaded into the memory of the secure cryptographic module, which then performs all the signing operations on behalf of the server. The keys are never revealed in their unencrypted form to the server, so even if an intruder manages to access the network, the keys will remain safe. Security is further assisted by physical design features of the module; tamper-resistant enclosures and advanced manufacturing techniques protect the keys from physical attack.

The signing of digital certificates is also a computation-intensive process, so it makes sense to consider combining some kind of hardware acceleration of cryptography within the key storage module. This way, keys are rapidly handled within a secure environment and no processing bottleneck is introduced, even when a high transaction throughput is required.

Selective encryption is the method of encrypting some parts of a compressed data file while leaving others unencrypted. Selective encryption is not a new idea. It has been proposed in various applications, especially in the commercial multimedia industry.

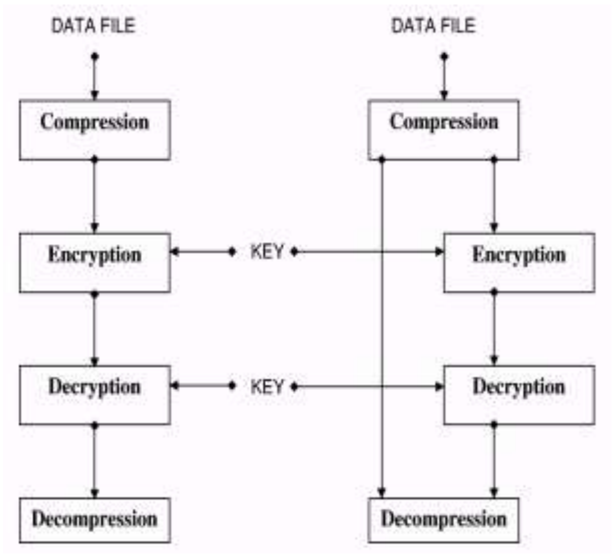
Hover, selective encryption of lossless compressed text files has not been explored, and that is the focus of this paper [3].

2. REVIEW WORKS

Huffman coding algorithm was introduced by David Huffman in 1952 and has been widely applied for data compression. The algorithm narrows the alphabet for the file based on the pattern of that particular data file and assigns the code for each character of the alphabet depending on the frequency of occurrence of that character. It assigns shorter code to the frequently used characters and longer code to the less-frequently used ones. Huffman coding therefore reduces the number of bits used for each high-frequency character while may increase this number for low-frequency character. These assignments results in the compression of the file to about 20 to 40%. Huffman coding therefore is fix-to-variable compression scheme.

As security is an increasing public concern these days, encryption is becoming popular for any type of sensitive information. An effective encryption scheme that saves the cost and time for data encrypting will be the need of the government, organizations, business companies or individuals. Selective encryption therefore has been proposed for this purpose.

Selective encryption suggests the technique that selectively encrypts just some parts of the compressed file while guaranteeing the security of the original data file. Our strategy for selective encryption here will nest the encrypting process into the encoding process while compressing a data file. With this arrangement, we are not only saving the time for encrypting the file, but also the cost of system complexity. The following figures explain the concept of selective encryption.



The Data Encryption Standard (DES) algorithm was adopted by the U.S. government in July 1977. It was reaffirmed in 1983, 1988, and 1993. DES is a block cipher that transforms 64-bit data blocks under a 56-bit secret key, by means of permutation and substitution. It is officially described in FIPS PUB 46.

DES is a "symmetrical" encryption algorithm: same key that is used for encryption is used to decrypt the message.

The DES algorithm is still widely used and is considered reasonably secure. There is no feasible way to break DES as is using a 64-bit (8 characters) block cipher. There are 70,000,000,000,000,000 (seventy quadrillion) possible keys of 56 bits. However, due to the advance in the computational power of super- computers, an exhaustive search of 2^55 steps on average, can retrieve the key used in the encryption (if the key is changed frequently, the risk of this event is greatly diminished).

3. PROGRAMS FOR ENCRYPTION AND DECRYPTION

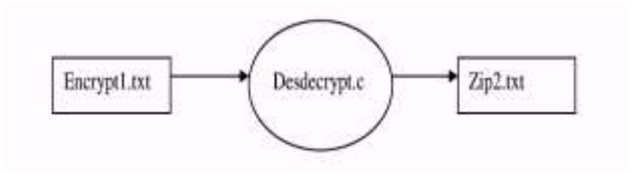
The program for Encryption and Decryption are presented below:

```
/* Program for Encryption */
#include<stdio.h>
#include<conio.h>
#define ISSET(a,b) a & pow(2,b) ? 1 : 0
char plain[8];
char cipher[8];
char key[8];
void pbox(char *, int, int *);
void sbox(char *);
void breakplain(char *, char *,char *);
void copytoleft(char *,char *);
void copytoright(char *,char *);
void findkey(char *,int);
void xor(char *,char *);
void display(char *);
void encrypt(void);
main()
{
        FILE *src,*dest,*skey;
        int n,i;
        clrscr();
        skey=fopen("d:/server/rsa_serv/skey.txt","r");
        fscanf(skey,"%s",&key[8]);
        fclose(skey);
        src=fopen("d:/server/zip1.txt","rb");
        dest=fopen("d:/server/encrypt1.txt","wb");
        while(1)
        {
                n=fread(plain,sizeof(char),8,src);
                if(n<8)
                        for(i=n;i<8;i++)
                        plain[i]='\0';
                encrypt();
                fwrite(plain,sizeof(char),8,dest);
                getch();
                if(n<8)
                        break;
        }
        fcloseall();
        getch();
        return 0;
}
void encrypt(void)
{
        int lkey[] = {1,5,4,2,6,7,0,3};
        int rlkey[] = {6,0,3,7,2,1,4,5};
        int skey[] = {1,3,0,2};
        char tempkey[4],left[4],right[4];
        int i;
        /* 1st Step */
        pbox(plain,8,lkey);
        /* 2nd to 17th Step*/
        for(i=0;i<16;i++)
        {
                breakplain(left,right,plain);
                copytoleft(plain,right);
                findkey(tempkey,i);
                xor(right,tempkey);
                sbox(right);
                pbox(right,4,skey);
                xor(right,left);
                copytoright(plain,right);
        }
        /* 18th Step*/
        breakplain(left,right,plain);
        copytoleft(plain, right);
        copytoright(plain,left);
        /* 19th step*/
        pbox(plain,8,rlkey);
}
```

```
void pbox(char *arr, int n, int *key)
{
        int i;
`       char *dest=(char*)malloc(n*sizeof(char));
        for(i = 0; i < n; i++)
                dest[key[i]] = arr[i];
        for(i=0; i < n; i++)
                arr[i] = dest[i]; return;
}
void sbox(char *arr)
{
        int i;
        for(i = 0; i < 4; i++)
                arr[i] = ~arr[i];
        return;
}

void breakplain(char *left, char *right, char *plain)
{
        int i;
        for(i=0;i<4;i++)
                right[i] = plain[i];
        for(i=4;i<8;i++)
                left[i-4] = plain[i];
        return;
}
void copytoleft(char *dest, char *src)
{
        int i;
        for(i = 3; i >= 0; i--)
                dest[i+4] = src[i];
        return;
}
void copytoright(char *dest, char *src)
{
        int i;
        for(i = 0; i < 4; i++)
                dest[i] = src[i];
        return;
}
int pow(int a,int b)
{
        int i,temp = 1;
        for(i = 1; i <= b; i++)
                temp *= a;
        return (temp);
}
void findkey(char *tkey, int n)
{
        int combine[] ={60,27,43,23,54,75,90,39,92
                        ,46,102,30,105,71,120,15    };
        int keyrule[] = { 2,0,3,1};
        int i,count;
        for(i=0,count = 0; i < 7 && count < 4; i++)
                if(ISSET(combine[n],i))
                        tkey[count++] = key[i];
        pbox(tkey,4,keyrule);
        return;
}
void xor(char *dest, char *src)
{
        int i;
        for(i = 0;i< 4;i++)
                dest[i] ^= src[i];
        return;
}
```

After the socket operations of two machines, the compressed, encrypted file (encrypt1.txt) has to be decrypted in the client machine. To do this, we use the DES decryption algorithm .This is just the reverse of the encryption algorithm. The output of the module is the compressed file.



The same algorithm can be used for encryption or decryption. In order to decrypt the cipher text and get the original text again, the procedure is simply repeated but the sub keys are applied in reverse order, from K16-K1. Other than that, decryption is performed exactly the same as encryption.

4. PROGRAM FOR DECRYPTION

```
/* Program for Decryption */
#include<stdio.h>
#include<conio.h>
#define ISSET(a,b) (a & pow(2,b)) ? 1 : 0
char plain[8], key[8];
void pbox(char *,int,int *);void sbox(char *);
void breakplain(char *, char *,char *);
void copytoleft(char *,char *);void copytoright(char *,char *);
void findkey(char *n,int);
void xor(char *,char *);void display(char *);void decrypt(void);
main()
{
        FILE *src,*dest,*skey;
        int n;
        clrscr();
        skey=fopen("c:/client/rsa_clit/skey.txt","r");
        fscanf(skey,"%s",&key[8]);
        fclose(skey);
        src=fopen("c:/client/encrypt1.txt","rb");
        dest=fopen("c:/client/zip2.txt","wb");
        while(1)
        {
                fread(plain,sizeof(char),8,src);
                if(feof(src))
                        break;
                decrypt();
                fwrite(plain,sizeof(char),8,dest);
        }
        fcloseall();
        return 0;
}
void decrypt(void)
{
        int lkey[] = {1,5,4,2,6,7,0,3};
        int rlkey[] = {6,0,3,7,2,1,4,5};
        int skey[] = {1,3,0,2};
        char tempkey[4];
        char left[4],right[4];
        int i,j;
        /* 1st Stage */
        pbox(plain,8,lkey);
        /* 2nd Stage */
        breakplain(left,right,plain);
        copytoleft(plain,right);
        copytoright(plain,left);
        /* 3rd stage to 18th Stage*/
        for(i=0;i<16;i++)
        {
                breakplain(left,right,plain);
                copytoright(plain,left);
                findkey(tempkey,15-i);
                xor(left,tempkey);
                sbox(left);
                pbox(left,4,skey);
                xor(left,right);
                copytoleft(plain,left);
        }
        /* 19th stage*/
        pbox(plain,8,rlkey);
        return;
}
void pbox(char *arr, int n, int *key)
{
        int i;
        char  *dest=(char*)malloc(n*sizeof(char));
        for(i = 0; i < n; i++)
                dest[key[i]] = arr[i];
        for(i=0; i < n; i++)
                arr[i] = dest[i];
        return;
}
void sbox(char *arr)
{
        int i;
        for(i = 0; i < 4; i++)
                arr[i] = ~arr[i];
        return;
}
void breakplain(char left, char *right, char *plain)
{
        int i;
        for(i=0;i<4;i++)
                right[i] = plain[i];
        for(;i<8;i++)
                left[i-4] = plain[i];
        return;
}
void copytoleft(char *dest, char *src)
{
        int i;
        for(i = 3; i >= 0; i--)
                dest[i+4] = src[i];
        return;
}
void copytoright(char *dest, char *src)
{
        int i;
        for(i = 0; i < 4; i++)
                dest[i] = src[i];
        return;
}
int pow(int a,int b)
{
        int i,temp = 1;
        for(i = 1; i <= b; i++)
                temp *= a;
        return (temp);
}
void findkey(char *tkey,int n)
{
        int combine[] = { 60,27,43,23,54,75,90,39,92,
                46,102,30,105,71,120,15};
        int keyrule[] = { 2,0,3,1};
        int i,count;
        for(i=0,count = 0; i < 7 && count < 4; i++)
                if(ISSET(combine[n],i))
                        tkey[count++] = key[i];
        pbox(tkey,4,keyrule);
        return;
}
void xor(char *dest, char *src)
{
        int i;
        for(i = 0; i < 4; i++)
                dest[i] ^= src[i];
        return;
}
```

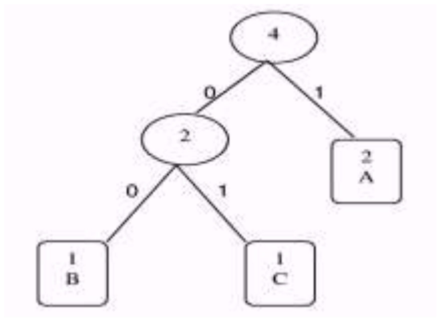
5. EXPERIMENTS

5.1. The simple cases

First experiments are with the simple case of 50% of 'A', 25% of 'B' and 25% of 'C'. The Huffman tree in this case would be as in the following figure below.

The string "AABBC", for example, would be encoded as "00101011".

Expectation for an effective system was type I system because type I system would affect all kinds of characters, while type II system would not affect the highest-probability character. For this case, the Huffman tree for the flipped bits with type I system would be:

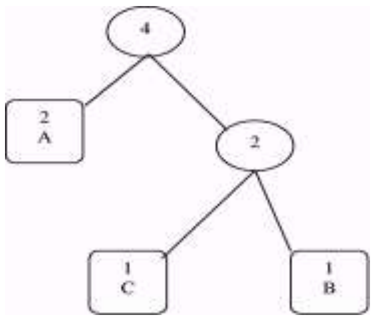


Char	Codeword
A	1
B	00
C	01

The string "AABBC" is now encoded and encrypted as "11000001".

It is therefore decoded (without being encrypted) as "CAAAAAA". The last "1" is left over because there is no such code of only "1". DSID in this case is 5.

With type II system, the Huffman tree for the flipped bits with type I system would be:



Char	Codeword
A	1
B	11
C	10

(Here also left 0 right 1)

The above string is now encoded and encrypted as "00111110".

Note that code word for characters "B" and "C" are swapped with each other and so "B" and "C" are swapped when they are decoded (without being encrypted). That is "AACCB" and has DSID of 3 which is 2 smaller than this value in type-l case.

The above string is now encoded and encrypted as "00111110". Note that code word for characters "B" and "C" are swapped with each other and so "B" and "C" are swapped when they are decoded (without being encrypted). That is "AACCB" and has DSID of 3 which is 2 smaller than this value in type-l case. But when we experimented with large number of characters (1000 characters, as we did), the complexity in the alignments of characters made DSID different from our anticipations.

6. CONCLUSIONS

Security has long been seen as a major adoption of Internet technology in the enterprise. As networks have grown and connected to the Internet, the spectre of the hacker has responsible for both delivering information within the enterprise and to its partners, and protecting it from unauthorised outsiders. The security capabilities of the latest Internet and intranet technologies enable companies to control the availability of information and the authenticity of that information better than ever before. The increasing sophistication of both server and client software means that this unprecedented level of security can be provided without requiring users to undergo complex and bureaucratic procedures to gain legitimate access to sites. This paper provides an outlook of network security and in the real text cases, the results were encouraging. With the ratio of encryption of about 10%, we could achieve the ratio of damage to the file of nearly 80% in some cases. Although we could not give a conclusion based upon those cases, they promised the potential of selective encryption with the ratio of encryption as low as 10%.

7. REFERENCES

[1] R.T. Morris, 1985. A Weakness in the 4.2BSD UNIX TCP/IP Software. Computing Science Technical Report No. 117, AT&T Bell Laboratories, Murray Hill, New Jersey.

[2] S.M. Bellovin. Security Problems in the TCP/IP Protocol Suite. Computer Communication Review, Vol. 19, No. 2, pp. 32-48, April 1989.

[3] Balachander Krishnamurthy and Craig E. Wills. Characterizing privacy in online social networks. In Proceedings of the Workshop on Online Social Networks, pages 37–42, Seattle, WA USA, August 2008. ACM.

[4] Information Security Forum. Information Security Standards. London: Information Security Forum, September 2001. p.1

[5] Vance, Bill. "Employees are your greatest assets… in security too!" March 2001.

www.techxans.org/resources/techxans.ppt

[6] Schneier, Bruce. Secrets and Lies - Digital Security in a Networked World. New York:

JohnWiley & Sons, August 2000. p.135-187, 255-317, 367-388

[7] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding", IBM

Systems Journal, Vol. 35, 1996, pp. 313–336.