2012 IACSIT Hong Kong Conferences
IPCSIT vol. 29 (2012) © (2012) IACSIT Press, Singapore

New Implimentation of Hashing and Encoding in Digital Signature

ERFANEH NOROOZI [1+], SALWANI BT MOHD DAUD [2], ALI SABOUHI [3], and MOHAMADREZA SALEH NAMADI [4]

[12] Advanced Informatics School (AIS), Universiti Teknologi Malaysia, Kuala Lumpur
[3] Software engineering of computer, Malaysia, Kuala Lumpur
[4] Doctor of Computer engineering, Tehran Jonob Branch, Tehran, Iran

Abstract. The purpose of introducing of this algorithm is a new method for designing a simple mechanism for producing a digital signature. Some applications like multi agent systems transfer messages with low size and capacity. The new algorithm minimizes the size of original file and gives us a dynamic and smaller size output. In this algorithm read the input file then hash the message and encode it. Finally modify the established code into a unique ID at Base 16. We concentrate on designing and implementation of functions of algorithm.

Keywords: Digital Signature, Symbolized Message, verification and validation.

1. Introduction

Most of the previous related works gave less attention on size of the hashed file and simplicity of generating digital signature appended at the end of message. Some applications need to have smaller size and simple and fast generation of signature as well as dynamic hashed file varies by the size of message. However, in electronic transferring, information is disposed to new and potential danger of security threats. In contrary to printed information on paper, information on electronic form can be stolen from far, hence main device for securing and establishing data is required. Digital signature is produced by machine. It produces a sequel regarding the arrival that is available by signer. A hidden entrance of the machine is needed so that no one is able to produce this sequel. Therefore, digital signature systems can be a solution to provide a secure method for transition of signed data using public key.

2. Digital Signature

Public-private keys are used to pass sensitive information however it can also be used to provide authentication on the particulars of a sender. It does not protect the contents of the message; it only provides particulars about the sender and source of the information. It provides authentication and integrity but does not provide confidentiality, data is sent as normal but acts like a normal signature we use on a letter. A digital signature works by creating a message digest which ranges from between a 128-bit and a 256-bit number which is generated by running the entire message through a hash algorithm. This generated number is then encrypted with the sender's private key and added to the end of the message.

When the recipient receives the message they run the message through the same hash algorithm and generate the message digest number. Next, they decrypt the signature using the sender's public key and providing the two numbers match they know the message is from who it says it's from AND that is has not been modified. A digital signature is basically a way to ensure that an electronic document (e-mail, spreadsheet, text file, etc) is authentic, Authentic means that you know who created the document and you

[+] Corresponding author.
E-mail address: ErfanehNoorouzi@iausepidan.ac.ir

know that it has not been altered in any way since that person created it. Digital signatures rely on certain types of encryption to ensure authentication. Encryption is the process of taking all the data that one computer is sending to another and encoding it into a form that only the other computer will be able to decode. Authentication is the process of verifying that information is coming from a trusted source. These two processes work hand in hand for digital signatures.

3. The implementation of suggestive algorithm

The goal of our algorithm is to propose a simpler mechanism to produce a digital signature. Our proposed algorithm introduces a simple mechanism for hashing and abstracting the message and a new technique in coding the abstract message to facilitate generating a unique index for each input file. The most important function in this algorithm is the hashing function.

First, we open the file with “w + t”, because “w” is the sign for write and “t” is the initial letter of text so we insert the data as a text and use “+” in order to show that the file is over write, so with this entrance of file, the previous file will deleted.

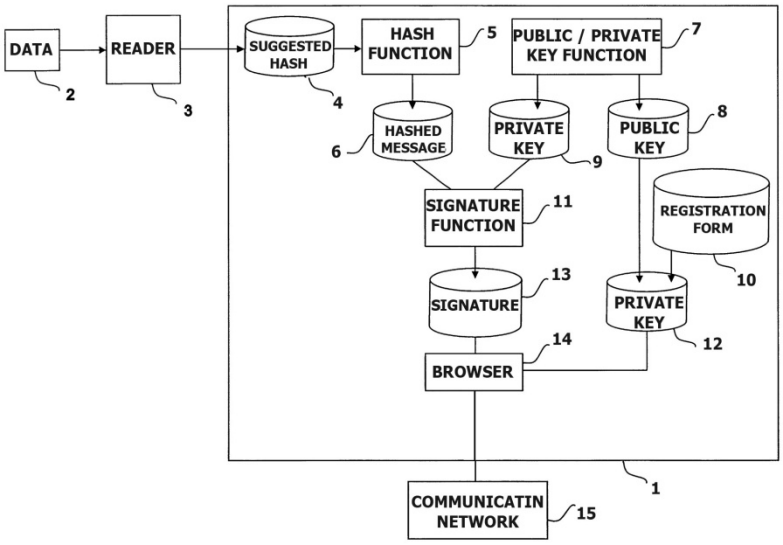

Figure 1.Summary of hashing and encryption in suggestive algorithm

3.1. Hash Function

In the proposed algorithm at first we did operation on first 100 bytes; the first byte is read by fetch function and placed it in variable character. If the rate of loaded bytes equals with zero we modified it to 1 because if the rate of one of them be equal with zero the final output won’t be zero. So it started from the first byte and before the end of the process, the next flowing operations begun.

We placed the loaded bytes in 32 variable bits and read the second byte from 100 bytes block. If this is the last one, we put the result of process in 32 bytes and modified it in hexadecimal and displayed it. If the next byte is not the last one, we multiply the result in 32 variable bits now if the result is less than 32 bits, then we add zero from the left side to fill it in 4 bytes. If the result of multiply is more than 4 bytes, we must keep the 32 bits lowest of the result into variable of char. The same process was repeated till the last byte.

```
void hashing(String Hash Filename)
{///Open the file, hash it and append at the end of file
unsigned long int l=0;
FILE *myhash;
myhash = fopen(hashfilename,"w+t");
while (!FEOF (Myfile)
{
 if (i==hashlen) {
    fprintf(myhash,"%lX",l);
    i=0;
          c=fgetc(myfile);
```

```
        c|=1;  };
    else {  l*=c;    i++; };
printf("\nAnalized Input File Was Compeleted To %s\n",hashfilename);
}
 fclose (Hash Filename);
  return 0 ;};
```

3.2. Encoding function

The encoder function reads the hashing file and starts the process of encoding, it opens hashing file with (Rb) r means read only and b means binary. Then open the file with "wt", "W" means write and t means text. We consider a 16 bytes private key and XOR the first byte of hashing file with first private key and put the result in first key byte. If we don't reach to the end of hashed file, XOR it with the first key and then put it inside second byte of the key. Once we don't reach to the end of hashed file XOR it with the first key and put it inside second byte of key. Next, we put the first byte of the key inside the hashed file and continued to the end of hashed file. We need to do the procedure because we want to modify the hashed file and repeat the iteration to the end of the key. Then we do the same operation for the second byte of the hashed message ($H[1]$). These iterations are done until the end of hashed message. Finally, a 16 byte key is appended at the end of message. The results of these operations were kept in a character array.

```
Void Encoder (String Hash Filename, string sign filename)
{//Encode the hash file with private key
        Fileopen (Hash Filename);
        Fileopen (Sign filename);
        While (! feof (hashfilename)
        {
      c=fgetc (Hash Filename);
      For (i=0; i<passlen – 1; i++)
          {  Password [i] =c ^ password [i];
           }       }
        Append File (password, Message File);
        CloseFile (Sign filename);
        CloseFile (Hash Filename);}
```

Hashed file is named as sign file and stored in the as text format in drive C as hashed file. During operation, the "signature version" appeared on the screen, and then the message 'enter file name' will be shown. The user is required to enter the path of file and also the name of file and p suffix of PDF and opens file with rb (only read binary). The hashing and encoder function is finally fetched during conversion to hex format. A unique code for each file is created. So when there is a change in the first file, for example if one character will be omitted or the output hash of function and output of sign function will differ from the first file.

```
int main(void){    int x,y;    int mn;
strnset(sign,' ',passlen*2);
FILE *myfile;
char filename[80];
textmode(C80);    clrscr();
printf("Singniture Generator Version 1.00\n");
printf("Enter File Name : ");
 scanf("%s",filename);
myfile = fopen(filename, "rb");
if (myfile==NULL)    {
printf("\nCan not Open File %s !!!.\n",filename);
exit(0);  }
else    {
hashing(myfile);
printf("\nHashing Was Compeleted ...\n");
```

```
encoder();
printf("Result Was : %s\n",sign);
fclose(myfile);
getch();    getch();  return  0;};
```

3.3. Convert to hexadecimal function

These function intakes an array of char, and modifies it to hex a decimal. We introduce 2 variables Lo, hi and we do this operation from the beginning till the end of encoded file

Hi=data [i] &240

Lo=data [1] & 15

We use Hi 8 Lo because 240 in binary system equals (11110000) 15 in binary system equals (….1111) and hi>>4 in variable of Hi equals with this sentence "shift Hi 4bits to right" so Hi changes to (00001111) and is similar to 10 in appearance.

Now for creating the code in Hexa decimal if (hi>9) then we find substance of 'A' to zero and then add the result to hi=hi+'A'-'0' and if (L0>9) we substance of 'A' to 9 from one and add it to lo=lo+'A'-'9'-1 from main function. There are 2 arrays called passlen with the primary size of 16 and hashLen with the primary size of 100, because we read 100 bytes of data from a file then we consider the variable of password as 16 bytes and also the length of it equal passlen and finally produce 32 bytes variable as key signature.

```
int convert2hex(char*data,char*result)
{    int j=0;    int i=0;    int hi,lo;
 for (;i<passlen;i++)    {
   hi=data[i] & 240;

  lo=data[i] & 15;
   hi=hi>>4;
   if (hi>9)     {       hi=hi+'A'-'0';    };
   if (lo>9)     {       lo=lo+'A'-'9'-1;    };

hi+='0';
lo+='0';
   result[j++]=hi;
 result[j++]=lo;    };  return 0;};
```

4. Conclusion

A digital signature is computed using a set of rules and a set of parameters such that the identity of the signatory and integrity of the data can be verified. A hash function is used in the signature generation process to obtain a condensed version of data.Partial change in the place of text or any attack will cause big change in the final file of digital signature.

In this paper, we introduced a new digital signature algorithm which generates the hashed file with dynamic size that is the hash function result depends on size of message. Our algorithm introduces a more simple mechanism for hashing and encryption. The functional and time complexity of our algorithms is better than the others. Our algorithm works with all types of files such as .doc, .pdf, .txt and other types of files. Our proposed algorithm is applicable for applications that send messages with low size such as multi agent systems in which agents send messages in a few bytes. Our future work is to evaluate the effectiveness of the proposed algorithm under other attacks. We want to develop a model check using an actor based language such as Rebecca.

5. Acknowledgements

We would like to express our gratitude to Advanced Informatics School of Universiti Teknologi Malaysia for realizing this research work. Our special thanks to Ministry of Higher Education (MOHE), Malaysia for their support in providing research grant for this study.

6. References

[1] Y.–M. Tseng, "*Cryptanalysis and restriction of an automatic signature scheme in distributed systems*," IEICE Transactions on Communications, Vol. E 86 -B No. 5 , pp. 1679 - 1681 , May 2000.

[2] K. Usuda, M. Mambo, T. Uyematsu, and E.Okamoto, "*Proposal of an automatic signature scheme using a compiler*," IEICE Transactions Fundamentals, Vol. E 79-A, No. 1, pp. 94- 101 , January 2004 .

[3] DE. Denning," *Cryptography and Data Security* ", Addison -Wesley Publishing Co., July 2003.

[4] R.M. Needham andM.D.Schroeder,"*Using Encryption for Authentication in Large Networks of Computers*," Vol. 21 , pp. 993-999 . , Dec 2007 .

[5] J. Linn, "*Practical Authentication for Distributed omputing*", Proc. IEEE Symp. Research in Security and rivacy, pp 31 - 40 , Apr 2005.

[6] M. Gasser and E. McDermott, "*An Architecture for Practical Delegation in a Distributed System*", No. 2060 , pp 20 - 30. July 1999.

[7] M. Gasser, "*The Digital Distributed System Security Architecture*",pp 305 - 319 Dec 2002 .

[8] J.G. Steiner and J.I. Schiller," *An Authentication Service for Open Network Systems*", Proc.Winter Usenix Conf , pp 191 - 202 . Aug 1998.

[9] S.M. Bellovin and M. Merritt, "*Limitationsof the Kerberos Authentication System*", Calif , pp 253 - 257 . Oct 2004 .

[10] J.J. Tardo and K. Alagappan, *" Global authentication Using Public Key Certificates*", Order No. 2168 ,pp. 232-244 Dec 2003.