Proceedings of International Conference
on Cryptology and Information Security

A Software Implementation of Multisignature Scheme for Electronic Document Systems

Ki-Yuan Wang, To Chang, Erl-Huei Lu*, and Jau-Yien Lee*

Department of Electrical Engineering, Chung Cheng Institute of Technology

*Department of Electrical Engineering, Chang Gung College of Medicine and Technology

Abstract

The implementation of the security environment for electronic documents not only can improve the efficiency of document processing, but also can provide a secure and reliable environment. It is well known that the security environment for electronic document systems consists of many components, such as password authentication, multisignature, access control and key assignment, etc. And multisignature is the core of the security of such systems since it concerns the security of digital documents generation, signature authentication and safe storage of an electronic document. In this paper we implement a software multisignature system with C language for PCs which allows unpredetermined signing order for general electronic documents. We also provide test result for performance analysis.

Key words: multisignature, public-key cryptosystem, digital signature.

1. Introduction

The first public key cryptosystem concept was introduced by Diffie and Hellman [1] in 1976. Then, in 1978, Rivest, Shamir and Adleman[2] presented the first public key encryption scheme, which received the widest attention among several public key systems[2-5]. This scheme can be used for data encryption and digital signature for one-to-one communication. Few attempts has been made to apply RSA scheme to multisignature cryptosystems due to the problems of bit expansion and moduli size clashes for multisignature[6-7]. Though several resolutions have been proposed but still impractical, not to mention a software implementation of RSA cryptosystem for realistic electronic documents environments.

Recently, Chang, et al., [8] presented a multisignature scheme suitable for environments where multiple users involved in signing a document with random signing order. In their method each user possesses a set of modulo numbers n and corresponding secret and public keys which are delimitated by the increasing values H (see Fig. 1). It is this structure that provides the random signing order capability. Since their method is more practical than the others, we implement a software system based on that method which is suitable for electronic document systems. In our software system a one-way-hashed document digest is signed instead of a clear text, in order to reduce the complexity of calculating modular exponentiation, communication traffic, storage space and to avoid the known-signature attack.

We used a Chinese text file created with Microsoft Word 6.0™ as our test document and the test result is also provided for performance analysis.

This paper is organized as follows: Section II describes the system structure, system modules, users interface, files format, one-way hash function, modular exponentiation, functional modules and keys management of the multisignature system. In Section III we evaluate the system performance, and in Section IV, we cryptanalyze the system followed by a brief conclusions.

2. System design

2.1. System structures

The complexity of the system solely depends on the number of users. In the system, each user is assigned with same number of pairs of secret and public keys. These keys are delimitated with increasing values H as shown in Fig. 1. With such keys assignment, the bit expansion problem of modular exponentiation in RSA-based multisignature scheme[6] can be avoided, hence enable multiple users to sign on one document without predetermined order[8].

When a user enters the multisignature system he/she could be either the document initiator who creats a document to be signed by other users, or an intermediate verified receiver who authenticates the received signature then signs. Therefore, the comprehensive structure of the system is depicted in Fig. 2. In the system, signatures and the clear texts are send through e-mail to receivers.

Since the modular exponentiation is the most time-consuming operation in the RSA scheme and it's complexity is proportional to the length of the binary expression of key and modulo n. So, we use a one-way hash function to compress a clear text into a block of digest of fixed length. Then the complexity of space and time are confined to a constant and the software system is thus workable in a PC environment. Moreover, the known-signature attack can be avoided if a digest were signed instead of clear text.

Since the digest is signed and send to the next receiver, the authentication procedure requires that clear text is send together with the signature for receiver to verify. When the receiver receive clear text and signature, he/she also hashes the clear text and compare to the result from encrypting the signature with former user's public key. It is authenticated if they are the same.

2.2. System modules

From the discussions stated above, the complete system function is based on two functional modules, they are:1. a one-way hash function which compresses a variable-length text into a fixed-length digest, and 2. a multisignature algorithm. For 1, we adopted the Message Digest MD5[9] to hash a text into a digest. For 2, we applied Chang's et al improved RSA-based multisignature algorithm. Fig 3. shows functional blocks of these two modules.

2.3. User interfaces

A user interface composes of the functional menu a user can choose after he/she enters the multisignature system. System assigned each legal user an ID number and a certain number of pairs of keys for the functions of signatures and authentications. And the interfaces can be categorized as follows:

2.3.1. Interfaces for multisignature

When a user selects the function of signature, the system prompts a screen for user to input the name of file to be signed and the name of file to store the signed document. The system will also require the user to input his ID and the ID of next receiver to be verified and for the system to access corresponding public key files. The system now reads in the file named by the user but must first identify the contents of the file as either a clear text or a signature. If it is a signature then after authentication the user directly signs on the signature and the procedure of hashing can be omitted. But if it is a clear text then hashing procedure must be applied to transform the text into a 128-bit digest before it is signed.

To be sure that the contents of the file be identified correctly, we choose a identification value L=25, and let the length of the clear text be over 25 bytes, this is because after it is one-way hashed, the length is 128 bits (16 bytes). Then after the modular exponentiation the length remains the same. Now, we can tell it is a signature if the length of the file is less then L.

A signature selection menu consists of the following items:

I. input the name of the file to be signed
II. input the name of the file to store the signature
III. input signer's ID and receiver's ID

2.3.2. Interfaces for authentication

When a user selects the function of authentication, the system prompts a screen for user to input the name of clear text and the name of corresponding signature. In the system, the string of concatenated user's ID is stored together with signature as a header, therefore the system can access this string to identify the number of signatories and access their corresponding public keys to transform signatures back to digests with these public keys. And the signature is authenticated if the recovered digest and the one-way hashed digest are the same. The complete procedure of authentication is depicted in Fig. 4.

An authentication selection menu consists of the following items:

I. input the name of signature file
II. input the name of clear text file

2.4. File formats

An electronic document system is a system where digitalized files are generated, processed and stored. And it is these digitalized files what we defined a clear text and a signature in multisignature systems. No matter what the contents may be.

Fig. 5. shows the format of a signature file in the system which had been signed by four different users, the hexadecimal 20,3e is the delimitation to separate ID data from signature data. The length of each ID is temporarily set to 5 bits, it can be variable. And the length of signature is 16 or 17 bytes.

2.5. One-way hash function

The purpose of using one-way hash function to compress a message of variable length to a digest of fixed length is to reduce the complexity of the system. Here in this system, we applied MD5[9] to compress messages into digests of 128 bits (16 bytes).

In MD5, the file is padded so that its length is just 64 bits short of being a multiple of 512. This padding is a single "1" added to the end of the file, followed by as many "0"s as required. Then, a 64-bit representation of the length of the file (before padding bits were added) is appended to the result. These two steps serve to make the message length an exact multiple of 512 bits in length, and ensure that different files will not be the same after padding.

Therefore, in software implementation a block of 64 bytes (512 bits) of data is read in from the file. And MD5 processes this 512 bits block, divided into sixteen 32-bit sub-blocks. The output of the algorithm is a set of four 32-bit blocks, which concatenate to form a single 128-bit hash value. Thus, we define an array whose length is 64 bytes and each sub-array is 32-bit long, as the buffer to store the read in data.

Now, a padded clear text file can be divided into L sub-blocks, each with 512-bit as its length and are denoted as $Y_0, Y_1, \ldots, Y_{L-1}$. MD5 then processes these sub-blocks iteratively as shown in Fig. 6. For details of MD5 algorithm, please refer to [9].

2.6. Modular exponentiation

In the system, a 128-bit signature is generated from modular exponentiating a 128-bit hashed digest. Thus, the lengths of digest, keys, and modulo n are all of 128-bit. To compute the modular exponentiation on PC is time-consuming. So we applied Knuth's Binary Method[10] to reduce computational complexity.

It is well-known that Binary Method not only can reduce computational complexity of modular exponentiaiton but also can make the coding of software programs much easier. The fundamental algorithm for computing $A = M^d \bmod n$ is described as follows:

1.Let d be represented as a binary string, e.g., $d=(101011101.....)_2$.
2.Let $A=M$.
3.Scan d from left to right, start at the second bit
  i. if "1" then square A and mod(n), then times M and mod(n), then let A=result;
  ii. if "0", then square A and mod(n), then let A=result.

It also can be scanned from right to left[11]:

1.Let d be represented as a binary string, e.g., $d=(101011101.....)_2$.
2.Let $A=1$, and $B=M$.
3.Scan d from right to left
  i. if not the last bit then $B=B \times B \pmod n$;
  ii. if "1" then $A=A \times B \pmod n$.

Either scan from left to right or from right to left, the complexity of computing modular exponentiation is dramatically reduced to that of modular multiplication, i.e., computing $M \times M \pmod n$. But it is still time-consuming when M is large. Therefore we adopted an improved method in [11]:

Let E, M and n are binary integers of length l, where $0 \le M < n$. Let binary expression of M be

$$M = m_{l-1}2^{l-1} + \ldots + m_1 2 + m_0$$

where $m_i = 0$ or $1, 0 \le i \le l-1$.

Therefore $E=M \times M \bmod n$ can be expressed as

$$E = [(Mm_{l-1}2^{l-1}) + \ldots + (Mm_1 2) + (Mm_0)] \bmod n \quad (1)$$

Now, let $E_0=0$, then Eq(1) can be computed iteratively as follows:

First compute

$$E_1 = \left[\left(E_0 + Mm_{l-1}\right) \bmod n\right] 2$$

then

$$E_2 = \left[(E_1 + Mm_{l-2}) \bmod n\right]2$$
$$\vdots$$
$$E_{l-1} = \left[(E_{l-2} + Mm_0) \bmod n\right]2$$

after l steps we have

$$E_l = \left[(E_{l-1} + Mm_0) \bmod n\right] = E$$

It is obvious that this method needs only l modular additions insteat of l^2 modular multiplications in $M \times M \pmod n$. After each iteration of computing E_i, its range must be $0 \le E_i < Mm_{l-1-i}$.

2.7. Functional modules

From the design considerations and algorithms studies stated above, we defined functional modules and sub-modules as follows which are the core of the software implementation:

MD5 and MODEXP( ) are the two constructing functional modules of the software system.

For the module of MD5, it can be decomposed into three sub-modules, they are:

1.APPEND( ): to perform string padding of read in file, so that file length is congruent to 448 mod 512.

2.REG( ): to generate a 128-bit digest block. It consists of executing 16 times of ROUNDi( )[9].

3.ROUNDi( ): to perform ith round computation.

As functional module MODEXP( ) is concerned, it is clear that a modular exponentiation can be expressed as modular multiplication, and modular multiplication can be expressed as modular addition. Thus, a modular exponentiaiton can be decomposed into a addition operation and a modular operation. The sub-modules which constitute the MODEXP( ) are listed as follows:

1.COMPARE( ): to perform comparison of two integers.

2.SHIFTLEFT( ): to perform shift one bit position to the left.

3.SHIFTRIGHT( ): to perform shift one bit position to the right.

4.ADD( ): to perform the addition of two integers whose length are 128 bits.

5.SUB( ): to perform the subtraction of two integers whose length are 128 bits.

6.MOD( ): to perform modular operation of two integers whose length are 128 bits.

7.MODMUL( ): to perform modular multiplication.

2.8. Keys management

In the system, the generation of user's keys for multisignature must meet the requirements depicted in Fig. 1 so that the bit expansion problem can be avoided. We adopted key generation program from [12] to assign each user the certain number of keys as needed. These keys are stored as files depicted in Table 1, where files with extension .PUB indicates it is a public key file and .SCT indicates it is a secret key file.

In each .PUB files, it contains public key elements e and n, where e is a 32-bit long integer and n is a 129-bit long integer. In each .SCT files, it contains secret key elements (p,q,d,n), where p and q are prime numbers of 64-bit and 65-bit respectively, d is a 128-bit long integer, n is the same. All these public keys and secret keys are all stored as hexadecimal format.

These key files can be accessed for encryption or decryption according to the list of ID that is shown in the header of signature file (Fig. 5) which is transmitted among users. The .PUB files should be put into a directory for public access, and the .SCT files must be kept as user's personal secrets.

3. Performance analysis

As previously stated, we used MD5 to hash variable length messages into fixed length digests in order to reduce the inherited computational complexity in RSA cryptosystems. Therefore, for a single signature computation in our system, the total times required are the time for MD5 plus time for single (M,d,n) = (128,128,129) modular exponentiation. For the second user and the followings who are going to sign on the same signature, it only requires to compute a single (M,d,n) = 128,128,129) modular exponentiation. Table 2. shows the performances of time complexity of MD5 computation and modular exponentiation for a message and a signature, respectively.

We also evaluated the performance of signature of digests with different length, 128 bits, 256 bits and 512 bits,which results are shown in Table 3. The evaluation was conducted on a 486-DX66 PC with 8 MB RAM and 256KB CACHE.

From Table 3, we see that the time complexity increases proportional to the key length and modulo n. And according to [12], for a secure public-key cryptosystem, the length of keys and modulo n should be 512 bits at least. But in our system we still use 128 bits keys instead of longer ones in the light of system portability. Anyhow, the one-way hash function MD5 compensates the weakness of security from using short

keys.

4. Cryptanalysis

We intentionally skip the cryptanalysis of signature computation and the message recovery from signature since ther are the same as that in RSA cryptosystems. Here we cryptanalysis the possibility that an innocent user signs a forged signature from an adversary. In our system, any valid receiver will receive a pair of message $\{M, S\}$ for authentication where M is the clear text and S is the signature of M signed by the former users. If an adversary forged a clear text M', the hash value $h(M')$ of M' will not be equal to the hash value $h(M)$ of M. Besides, without user's secret key, the adversary can not generate a signture S* whose corresponding hash value $h(M^*) = S^{*e_i} \mod n_i$ will be the same as the forged hash value $h(M')$, i.e., $h(M^*)=h(M')$. Therefore, it is impossible that a valid user would sign on a message that is forged by an adversary.

Is it possible that an adversary can randomly selects two values M_1 and M_2, which satisfies $M_1 M_2 = M \mod n$ and send M_1 and M_2 to be signed by other users into S_1 and S_2 then forge a signature $S = S_1 S_2 \mod n$? Because

$$S = S_1 S_2 = M_1^d M_2^d = (M_1 M_2)^d = M^d \mod n .$$

The answer is no. In MD5, the procedure to hash a message into a digest goes through 4 rounds, 64 steps of nonlinear permutations. It is impossible that there exists a multiplicative function between $h(M_1)$ and $h(M_2)$. Moreover, a message is signed by a number of users in this system, it is impossible for an adversary to forge an intermediate signature to be verified as a legal one.

5. Conclusion

We presented a software system of multisignature which is coded in C language for electronic document environments. We analyzed and designed system modules as are one-way hash function and modular exponentiation, each consists of several independent sub-modules. We designed a user interface in DOS environment and tested its functions and evaluates complete system performance on a 486 PC. We also cryptanalyzed the system to assure its robustness in security.

Acknowledgment

The authors would like to thank Professor Chi-Sung Laih at Chen Kung University for the assistance in the coding of the software programs.

References

[1] Diffie, W. and Hellamn, M. E., "New Direction in Cryptography," *IEEE Transactions on Information Theory*, IT-22, pp. 644-654, 1976.

[2] Rivest, R. L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[3] ElGamal, T., "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, IT-31, pp. 469-472, 1985.

[4] Merkle, R., and Hellman, M., "Hiding information and signatures in trapdoor knapsacks," *IEEE Transactions on Information Theory*, IT-24, pp. 520-530, 1978.

[5] McEliece, R. J., "A public-key cryptosystem based on algebraic coding theory," *DSN Progress Report*, 42-44, pp. 114-116, 1978.

[6] Kiesler, T. and Harn, L., "RSA Blocking and Multisignature Schemes With No Bit Expansion," *Electronics Letters*, vol. 26, no. 18, pp. 1490-1491, 1990.

[7] Kohnfelder, L. M., "On the Signature Reblocking Problem in Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp.179, 1978.

[8] Chang, T., Pon, S.-F., Lu, E.-H., and Shyu, H.-C., " A Multisignature Scheme Without Predetermined Signing Order," *Journal of Chung Cheng Institute of Technology*, vol. 24, no. 2, pp. 169-175, Jan. 1996.

[9] Rivest, R. L., "The MD5 Message Digest Algorithm," RFC 1321, 1992.

[10] Knuth, D. E., The Art of Computer Programming, vol. II, Addison-Wesley, 1969.

[11] Lu, E.-H., "A Programmable VLSI Architecture for Computing Multiplication and Polynomial Evaluation Modulo a Positive Integer," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 1, pp. 204, Feb. 1988.

[12] Laih, C.-S., Harn, L., and Chang, C.-C., Contemporary Cryptography and Its Applications, UNALIS Co., 1995.

U_1	$n_{1,1}$		$n_{1,2}$		$n_{1,3}$		$\cdots$	$n_{1,w}$		$n_{1,w+1}$
U_2	$n_{2,1}$		$n_{2,2}$		$n_{2,3}$			$n_{2,w}$		$n_{2,w+1}$
U_3	$n_{3,1}$		$n_{3,2}$		$n_{3,3}$		$\cdots$	$n_{3,w}$		$n_{3,w+1}$
U_4	$n_{4,1}$	H_1	$n_{4,2}$	H_2	$n_{4,3}$	H_3		$n_{4,w}$	H_w	$n_{4,w+1}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\cdots$	$\vdots$		$\vdots$
U_t	$n_{t,1}$		$n_{t,2}$		$n_{t,3}$		$\cdots$	$n_{t,w}$		$n_{t,w+1}$

U_i : User's identity
H_i : Delimitation
$n_{i,j}$: User i's modulo number n at level j
Fig. 1. Structured chart of user's levelwise modulo numbers

Table 1. Examples of three levels keys files of user U_1

Names of public-key files	Names of secret-keys files
1-1.pub	1-1.sct
1-2.pub	1-2.sct
1-3.pub	1-3.sct

Table 2. Time complexities for MD5 and MODEXP( )

	MD5(seconds)	MODEXP( )(seconds)	Total(seconds)
Message	0.16	0.38	0.54
Signature	0	0.38	0.38

Notes: 1. the length of example message is 22,528 bytes.
2. the digest to be signed is 128 bits (hashed value of message).
3. d=128 bits, n=129 bits.

Table 3.MODEXP( ) times required for variable length of digests

PC models	(M,d,n) length (bits)	Time for $M^d \pmod n$ (seconds)
486DX-66	(128,128,129)	0.38
486DX-66	(256,256,257)	2.36
486DX-66	(512,513,513)	15.27

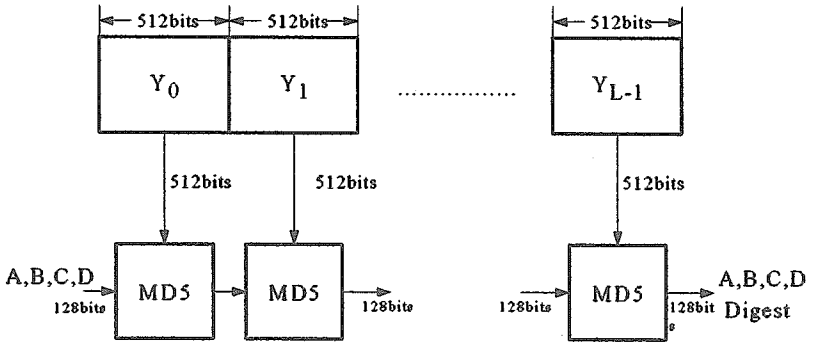

Fig. 6. MD5 generates the digest of a clear text

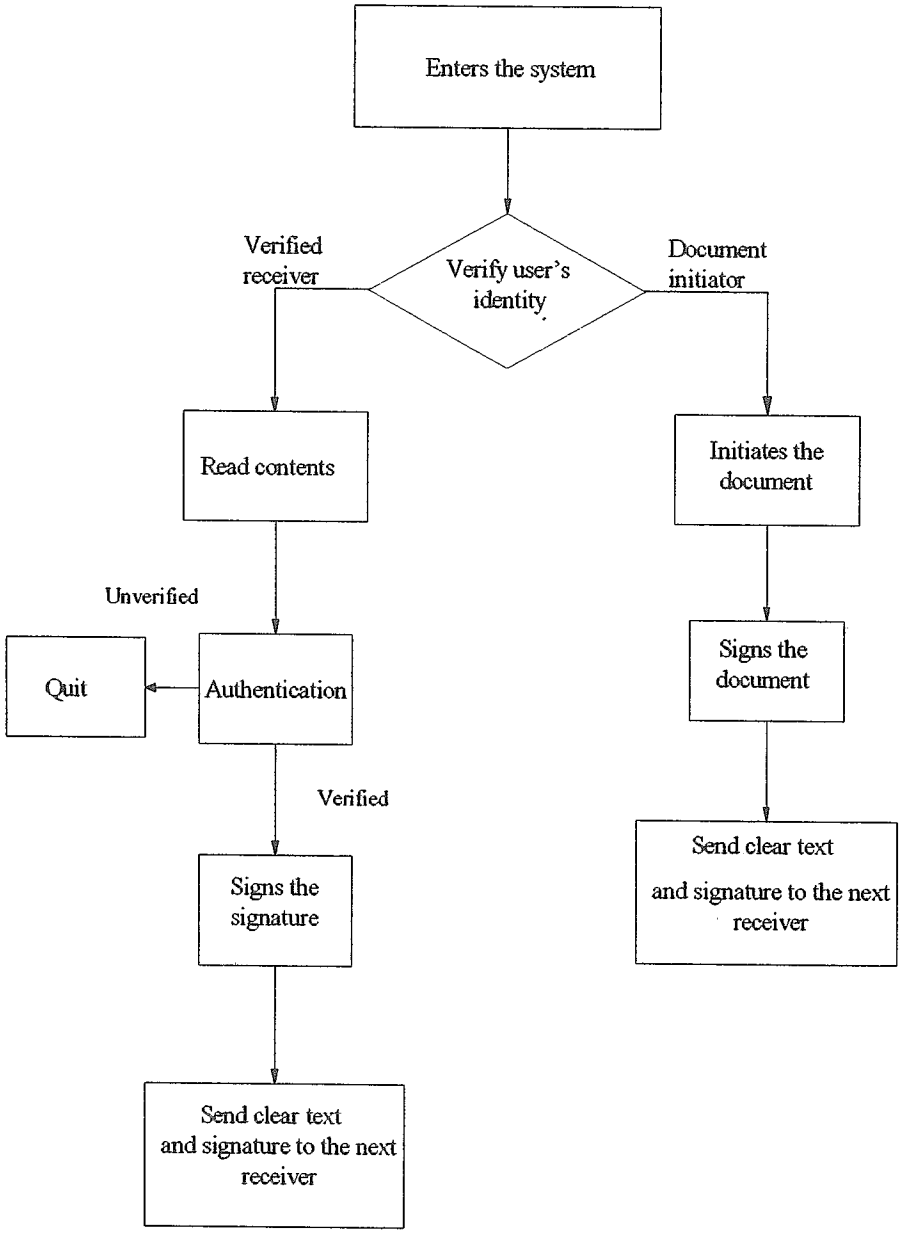

Fig. 2. System structure of multisignature software system

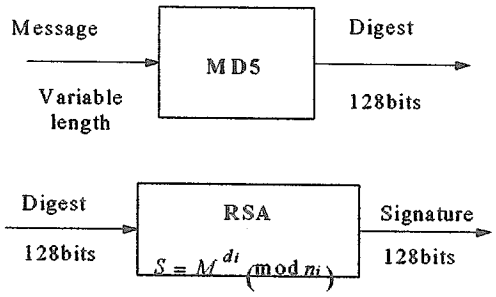

Fig. 3. Functional blocks of system modules

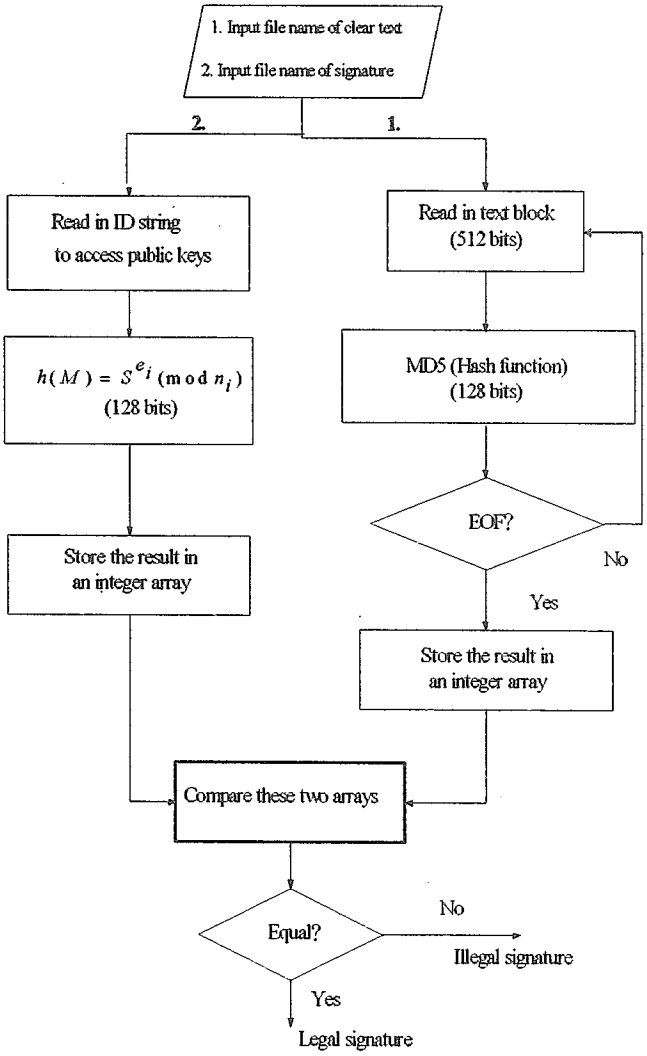

Fig. 4.Flow diagram of authentication procedure

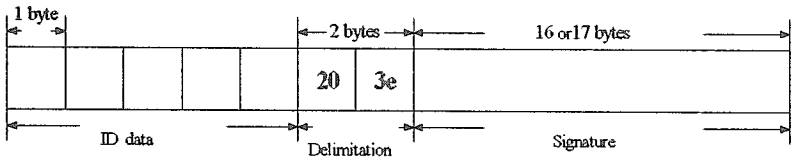

Fig.5. Format of signature files