Create A New Digital Signature Scheme Using Double Hush Function Algorithms

إنشاء طريقة توقيع رقمي جديدة باستخدام خوارزميات دوال تقطيع مزدوجة

Hameed Abdul-Kareem Younis* Israa Mahmood Hayder** Hussain A. Younis **

hameedalkinani2004@yahoo.com dd.swara@yahoo.com hussain394@yahoo.com

** Computer Science Dept., College of Science, University of Basrah, Basrah, IRAQ.*

*** Computer Science Dept., College of Science, Sam Higginbottom Institute of Agriculture, Technology & Sciences (Deemed to -be- University), Allahabad, Uttar Pradesh, India.*

Abstract

The rapid development of information and network technologies, networks already become an important part of peoples' live. People not only can get more information by net, but also can find new ways to live and new business work. For example, network bank, online chat, net-shopping, … etc. Along with the new things' appearance, the problem of information security becomes more important. Something more than authentication is needed. The most attractive solution to this problem is the digital signature which is analogous to the handwritten signature.

Digital signature technology more and more shows its important position in information security. The signature is formed by taking the hash of the message and encrypting the message. Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory. In addition, the recipient of signed data can use a digital signature as evidence in demonstrating to a third party that the signature was, in fact, generated by the claimed signatory. In this paper, an algorithm for signing text file has been presented. The famous RSA public key algorithm and the most popular hash function algorithms (SHA-1 & MD5) are used to generate the digital signature of a text file. The algorithm consist of two parts: signature generation and signature verification. Many experiments tested to examine the security of the presented work.

Keywords: Digital Signature, Hush Function, Rivest , Shamir and Aldman (RSA), SHA-1 & MD5.

الخلاصة:-

نظراً للتطور السريع في تكنولوجيا المعلومات والشبكات، أصبحت الشبكات جزءا هاما من حياة الناس . فالناس لا يحصلون فقط على معلومات من شبكة الانترنت، ولكن أيضا يمكن إيجاد طرق جديدة للعيش وأعمال جديدة. على سبيل المثال، الشبكات البنكية والمحادثة المباشرة، والتسوق من خلال النت ، ... الخ. جنبا إلى جنب مع ظهور أشياء جديدة أخرى، لذلك أصبحت مشكلة أمنية المعلومات أكثر أهمية. فأصبحت هناك حاجة إلى شئ موثوق أكثر من المصادقة وكان الحل الأكثر ملائمة لهذه المشكلة هو التوقيع الالكتروني الذي يماثل توقيع خط اليد.

تكنولوجيا التوقيع الرقمي تشكل جزء مهم من أمنية المعلومات. يتكون التوقيع من خلال اخذ ملخص الرسالة للرسالة المراد إرسالها ومن ثم تشفيرها. وتستخدم التواقيع الرقمية للكشف عن تعديلات غير مخول للبيانات والمصادقة على هوية الموقع. بالإضافة لذلك, مستلم البيانات الموقعة يمكن أن يستخدم التوقيع الرقمي كدليل في الاحتجاج إلى الطرف الثالث الذي عنة التوقيع بأنه ولد بالمولد المدعى. في هذا البحث، تم تقديم خوارزمية لتوقيع ملف نصي. يتم استخدام خوارزمية التشفير RSA الشهيرة ذات المفتاح العام وخوارزميات (SHA-1 و MD5) لإنشاء توقيع رقمي من ملف نصي. هذه الخوارزمية تتكون من جزئين: الأول لتوليد التوقيع والثاني للتحقق من صحة التوقيع. أجريت عدة تجارب لاختبار أمنية الخوارزمية.

1. Introduction

A digital signature algorithm authenticates the integrity of the signed data and the identity of the signatory. It may also be used in proving to a third party that data was actually signed by the generator of the signature. Is intended for use in electronic mail, electronic data interchange, software distribution, and other applications that require data integrity assurance and data origin authentication.

The digital signature is analogous to the handwritten signature. It must have the following properties:

- It must verify the author and the date and time of signature.
- It must authenticate the contents at the time of signature.
- It must be verifiable by third parties, to resolve disputes.
- Thus, the digital function includes the authentication function.

On the basis of these properties, we can formulate the following requirements for a digital signature:

- The signature must be a bit pattern that depends on the message of being signed.
- The signature must use some information unique to sender, to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new massage for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to return a copy of the digital signature in storage.

A secure hash function, embedded in a scheme such as that of figure satisfies these requirements. In this work, an implementation of the digital signature algorithm is proposed where a hash function, and a RSA algorithm is used.

2. Hash Function [1], [2]

A **hash function** is a (mathematical) function which receives an input of arbitrary length and returns a function value (hash value) of a fixed length (usually 128 or 160 bits). Its which are used in cryptography should be called *one-way hash functions.* The hash function itself must be public, i.e., everyone should be able to (efficiently) calculate the hash value for a given input. Conversely, however, it must be (computationally) infeasible to find an input for a given value which possesses exactly the predetermined value as hash value (this is referred to as a one-way characteristic). In cryptographic practice, generally hash functions must satisfy a second, more stringent requirement: it must be impossible to find two values which are mapped by the hash function onto the same hash value (collisions), since otherwise it would be possible to replace an already signed message after the event. Hash functions are especially used in connection with digital signatures. Here, for reasons of efficiency, the hash value of a message is signed instead of the message itself. The hash value H = H(M) of an original message M, is used to check, if the received message M' has been changed. In our proposal algorithm, we will used the most practical hash function algorithm (SHA-1& MD5). Now we will describe these algorithm in brief:

2.1 Secure Hash Algorithm (SHA-1) [3], [2]

The Secure Hash Algorithm (SHA) was developed by National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993; a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as (SHA-l). This algorithm has been well taken and appreciated by experts.

SHA-1 Logic

The algorithm takes as input a message with length of less than 2^{64} bit. And produces as output is 160-bit message digest. The input is processed in 512-bit blocks.

The algorithm steps are:

Step 1: Append padding bits. The message is padded so that its length is congruent to 448 modulo 512.

Step 2: Append length. A block of 64-bit is appended to the message. This block contain length of the original message.

Step 3: **Initialized buffer.** The buffer can be represented as five 32-bit registers (A, B,C, D, and E). These registers initialized with the following values:

A=67 45 23 01 D=10 32 54 76

B=ef cd ab 89 E=c3 d2 el f0

Step 4: **Process message in 512-bit blocks.** This step consist of four rounds of processing of 20 steps each. Figure (1) illustrate the logic of this step.

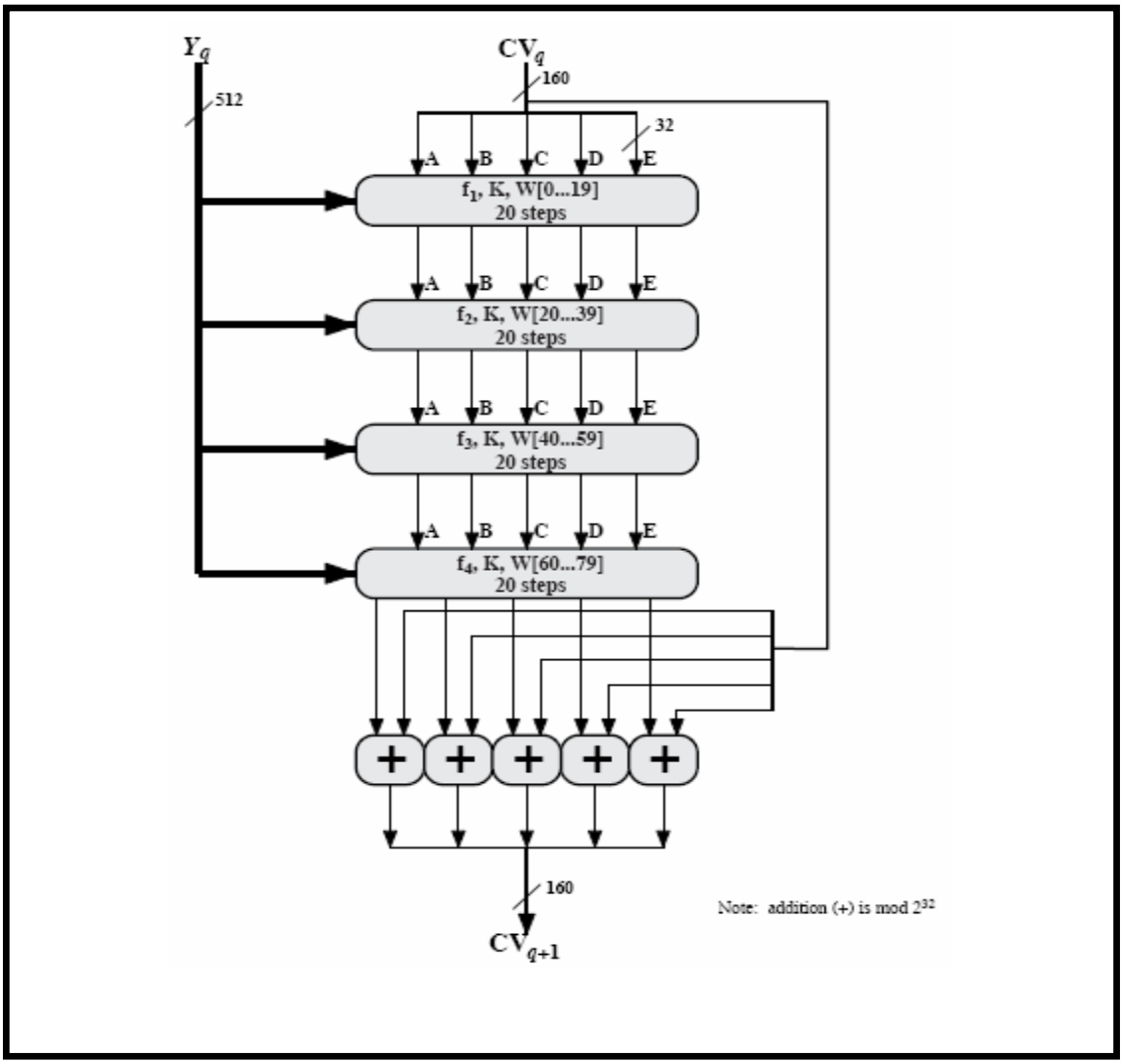


Figure (1): The SHA-1 compression function.

Where Yq: the current 512-bit being processed.

CV_q: 160-bit, the output of the previous block. CV_0=(ABCDE).

CV_{q+1}: 160-bit, the output of the current block.

W_t: a 32-bit word derived from the current 512-bit input block. The first 16 values of Wt are taken directly from the current block. The remaining values are defined as follows:

$$W_t = S^1 (W_{t-16} \text{ XOR } W_{t-14} \text{ XOR } W_{t-8} \text{ XOR } W_{t-3}) \qquad 16 \le t \le 79 \qquad \ldots(1)$$

S^k: rotation of the 32-bit argument by k-bits to the left. +: addition module 2^{32}.
K: constant. Table (1) explain K values.

Table (1): K values.

$0 \le t \le 19$	K_t=5A827999	$[2^{30} * \sqrt{2}]$
$20 \le t \le 39$	K_t=6ED9EBA1	$[2^{30} * \sqrt{3}]$
$40 \le t \le 59$	K_t=8F1BBCDC	$[2^{30} * \sqrt{5}]$
$60 \le t \le 79$	K_t=CA62C1D6	$[2^{30} * \sqrt{10}]$

Each steps from the 80 steps (4-round * 20-step) has the following form:

$$A,B,C,D,E \leftarrow (E+f(t,B,C,D)+S^5(A)+W_t+K_t),A, S^{30}(B),C,D) \qquad \ldots (2)$$

The four rounds have a similar structure, but different logical functions, which referred as f1, f2, f3, and f4**.** Table (2) explain the logic of the four functions.

Tab1e (2): Logic of SHA-1 functions.

Step	Function Name	Function value
$(0 \le t \le 10)$	f_1=f(t,B,C,D)	(B^C)OR(B'^D)
$(20 \le t \le 39)$	f_2=f(t,B,C,D)	B XOR C XOR D
$(40 \le t \le 59)$	f_3=f(t,B,C,D)	(B^C) OR (B^D) OR (C^D)
$(60 \le t \le 79)$	f4=f(t,B,C,D)	B XOR C XOR D

Step 5: Output. After all L 512-bit blocks have been processed, the output from the last stage is the 160-bit message digest (CV_L).

2.2 MD5 Message Digest Algorithm [2]

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^{128} operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort.

MD5 Logic

Step 1: Append Padding Bits. The message is "padded" (extended) so that its length (in bits) is congruent to 448 modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448 modulo 512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended [4] .

Step 2: Append Length. A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^{64}, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions). At this point, the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0... N-1] denote the words of the resulting message, where N is a multiple of 16.

Step 3: Initialize MD Buffer. A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first:

word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10

Step 4: Process Message in 16-word Blocks. We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word. Figure (2) illustrate the logic of this step.

F(X,Y,Z) = (XY) v (not(X) Z)
G(X,Y,Z) = (XZ) v (Y not(Z))
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))

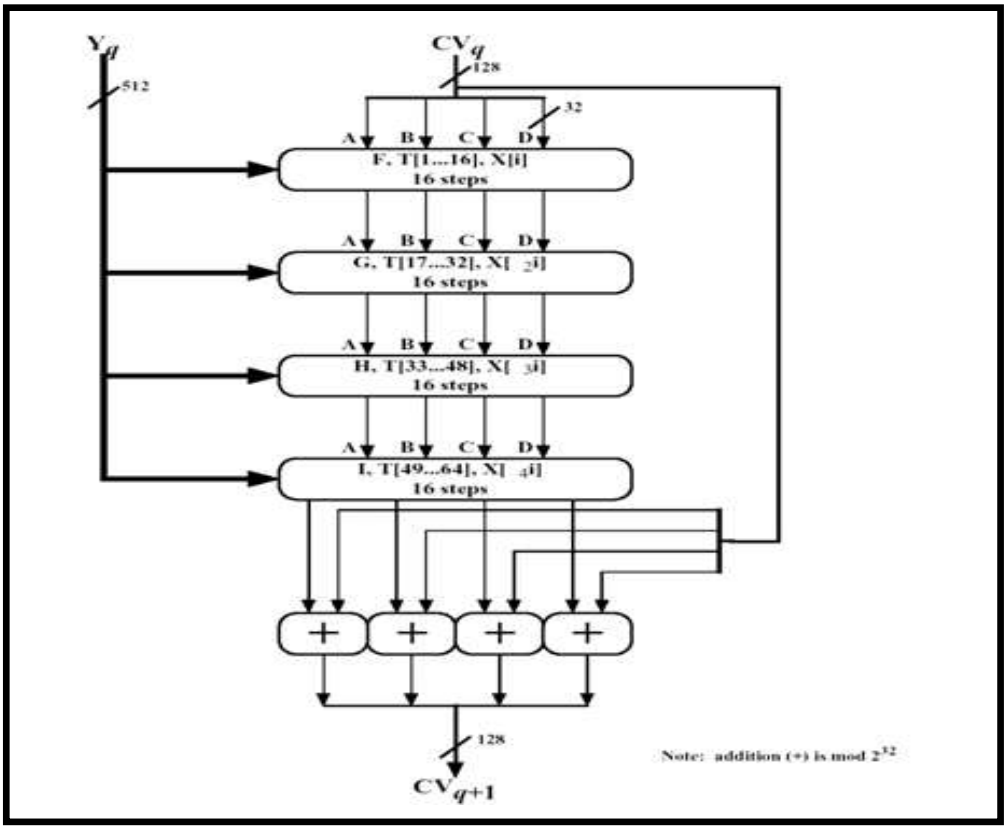


Figure (2): The MD5 compression function.

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position). It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, then each bit of F(X,Y,Z) will be independent and unbiased. The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs [5].

This step uses a 64-element table T[1... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians.

Do the following:

/* Process each 16-word block. */

For i = 0 to N/16-1 do /* Copy block i into X. */

For j = 0 to 15 do

Set X[j] to M[i*16+j].

end /* of loop on j */

/* Save A as AA, B as BB, C as CC, and D as DD. */

AA = A

BB = B

CC = C

DD = D

/* Round 1. */
/* Let [abcd k s i] denote the operation a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]
/* Round 2. */
/* Let [abcd k s i] denote the operation a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]
/* Round 3. */
/* Let [abcd k s t] denote the operation a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]
/* Round 4. */
/* Let [abcd k s t] denote the operation a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]
/* Then perform the following additions. (That is increment each of the four registers by the value it had before this block was started.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD
end /* of loop on i */

Step 5: Output. The message digest produced as output is A, B, C, and D. That is, we begin with the low-order byte of A, and end with the high-order byte of D. This completes the description of MD5.

2.3 The RSA Procedure [4]

As early as 1978, R. Rivest, A. Shamir, L. Adleman introduced the most important asymmetric cryptography procedure to date. It encrypt and decrypt numbers M less than n. The following algorithm explain the RSA work:

Key Generation

- Select two primes p,q, such that p≠ q.
- Calculate N= p*q.
- Calculate phi=(p1)*(q1). Phi: Euler phi function.
- Select integer e, such that: gcd (phi, e)=l; 1<e<phi.
- Calculate d, such that d ≡e1 mod phi.
- Public key {e, N}
- Private key = {d, N}

Encryption

Plaintext: M<n

Cipher text: $C=M^e \pmod N$

Decryption

Cipher text: C
Plaintext: $M=C^d \pmod N$

The size of an RSA key pair is commonly considered to be the length of the modulus *n* in bits. The corresponding RSA private key consists of the same modulus *N* and a private key exponent *d* that depends on *N* and the public key exponent *e(d is* the multiplicative inverse of e). Thus, the RSA private key is the pair of values (N, d) and is used to generate digital signatures. In order to provide security for the digital signature process, the two integers p and q, and the private key exponent d must be kept secret. The modulus N and the public key exponent e may be made known to anyone.

2.4 Signing with Hash Functions and RSA Encryption Algorithm

Rather than signing the actual document, the sender now first calculates the hash value of the message and signs this. The recipient also calculates the hash value of the message (the algorithm used must be known), then verifies whether the signature sent with the message is a correct signature of the hash value. If this is the case, the signature is verified to be correct. This means that the message is authentic, because we have assumed that knowledge of the public key does not enable you to derive the secret key. However, you would need this secret key to sign messages in another name. Some digital signature schemes are based on asymmetric encryption procedures, the most prominent example being the RSA system, which can be used for signing by performing the private key operation on the hash value of the document to be signed this scheme called *RSA digital signature*. The following figure explain the basic steps of RSA digital signature.

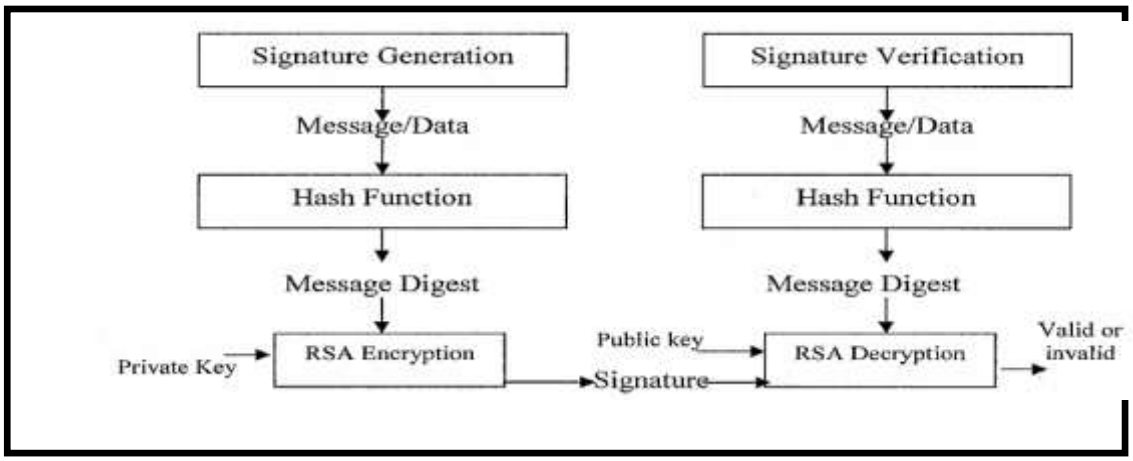


Figure (3): The basic steps of RSA digital Signature.

Our work consist of the following two algorithms:

2.4.1 Signature Generation

In this algorithm, the sender sign his message by using his private key and send it to the recipient as the following steps:

1-Message Choosing: Determine the message to sign it. Here, a text file has been as a message.

2-Hash Function: We use SHA-1 & MD5 algorithms to compress the message and compute the hash code H from each , hash code consist of 160-bit obtain from SHA-1 and 128-bit from MD5 . After that we take these hash codes and use XOR operator between this codes to produce hash code (H) of 160-bit

3- Key Generation: Choose two prime numbers p, q. Compute the RSA modulus N, such that N=p*q. Then, compute Euler phi function, where: phi=(p-1)*(q-1). Select the public key e, where gcd (e, phi)=1, and 1<e<phi. Finally, compute the secret key d which is the multiplicative inverse (e^{-1}) of e. The d value computed by using extended Euclid algorithm as the following [1]:

EXTENDED EUCLID(*phi*, *e*)
1. (A1, A2, A3)=(1, 0, *phi*);
(B1, B2, B3)=(0, 1, *e*)
2. if B3 = 0
return A3 = gcd(*phi*, *e*); no inverse
3. if B3 = 1
return B3 = gcd(*phi*, *e*); B2 = e^{-1} mod *phi*
4. Q = A3 div B3
5. (T1, T2, T3)=(A1 – Q B1, A2 – Q B2, A3 – Q B3)
6. (A1, A2, A3)=(B1, B2, B3)
7. (B1, B2, B3)=(T1, T2, T3)
8. goto 2

where d = B3.

4- Pre-encryption: In this step, we initialize the hash code by take the hash code (H) result from step one to produce vector VEC of 40 byte (in hexadecimal).Then we will split this vector to tow vectors each of them contain 20 byte and use the XOR operator between them to produce Signvector1 of 20 byte (in hexadecimal) then convert the values to decimal to produce Signvector of 10 byte.

5- Sign Computation: In this step, we use the private key (d) of the sender (signatory) to encrypt each number M in signvector as the following:

$C = M^d \bmod N$ . . . (3)

to obtain the 10 numbers sign vector (sign).

6-Sign Appending: The obtained sign is appended in the bottom of the signed text and send to the recipient of that text.

All these steps are shown in Figure (3).

2.4.2 Signature Verification

In this algorithm, the recipient do the steps to verify the sign in the received message by using the public key of the sender:

1-Signature Authentication: The received signature numbers are decrypted first by the public key (e) of the sender(signatory). As the following equation:

$M = C^e \bmod N$ …(4)

Then each obtained block is converted back to its original two bytes. .

2- Hash Code Comparison (Integrity): Compute the hash code of the received message (text) again and compare its hash code with the received hash code (result of step l). If the two codes are equivalent, then we obtain the following results:

a. The message has not been modified during the transmission, since its hash code equal the hash code of the signed message.

b. Message authentication, since its delivered by the public key of the sender.

c. No source-repudiation, since the public key of the sender used to produce the correct hash code for the received message. This figure bellow explain these steps of digital signature algorithm:

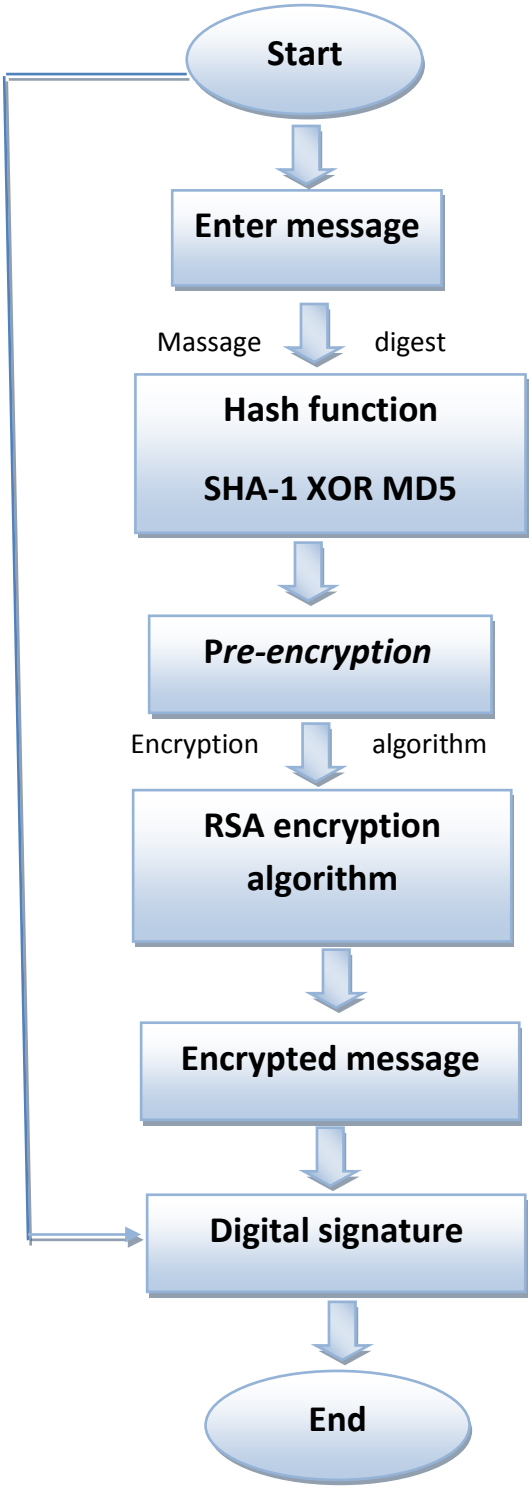
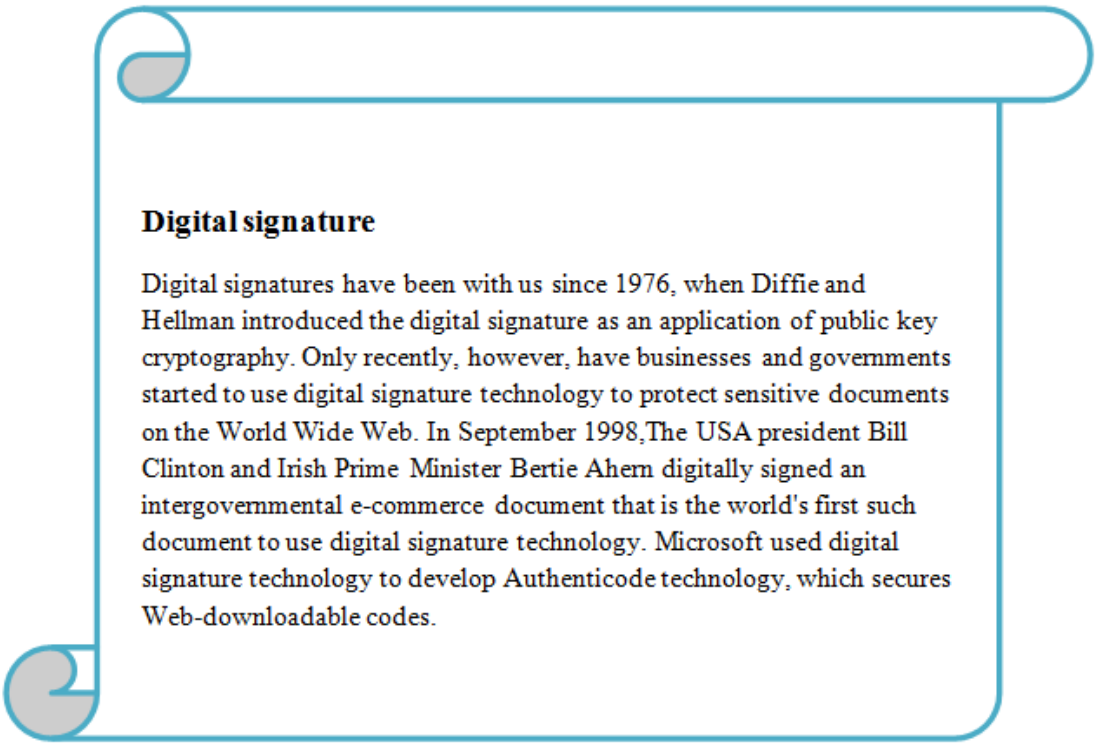


Figure (4): The digital signature algorithm.

3. Experimental Results

In the present work, different experiments has been tested to explain how can we sign a text file by RSA digital signature. We use the following text as a message to be signed:



Digital signature

Digital signatures have been with us since 1976, when Diffie and Hellman introduced the digital signature as an application of public key cryptography. Only recently, however, have businesses and governments started to use digital signature technology to protect sensitive documents on the World Wide Web. In September 1998,The USA president Bill Clinton and Irish Prime Minister Bertie Ahern digitally signed an intergovernmental e-commerce document that is the world's first such document to use digital signature technology. Microsoft used digital signature technology to develop Authenticode technology, which secures Web-downloadable codes.

Experiment 1

In this experiment, signature generation has been tested. First we compute the hash code H(1) for our text by using SHA-l hash algorithm. We obtain the following 160- bit:
H(1) = (90 2A 9A 99 D5 79 46 AC 63 26 1C D4 DE 12 BC 98 E2 57 FD 9F)Hex. And then we compute the hash code H(2) for the same text by using MD5 hash algorithm. We obtain the following 128- bit:
H(2) = (71 2D 78 E7 A1 D4 6E C4 62 79 82 1C DA 1A C9 1C) Hex . Then, we compute the hash code H by taking the two hash codes H(1) and H(2) and make the XOR operator between them we obtain the following 160- bit:
H=(6F D5 65 66 5B AB C1 D4 3D 0D 8D EF 43 94 C1 7D C4 32 CB 7C) Hex .
Next, we will generate the key that will be used in RSA algorithm following parameters for the key generation of RSA algorithm:
- p= 5419,q= 8681,
- Compute N=p*q= 5419* 8681=47042339.
- Phi =5418*8680= 47028240.
- Select the public key e=37.
- Compute the private key d by using extended Euclid algorithm. We obtain d=44486173.

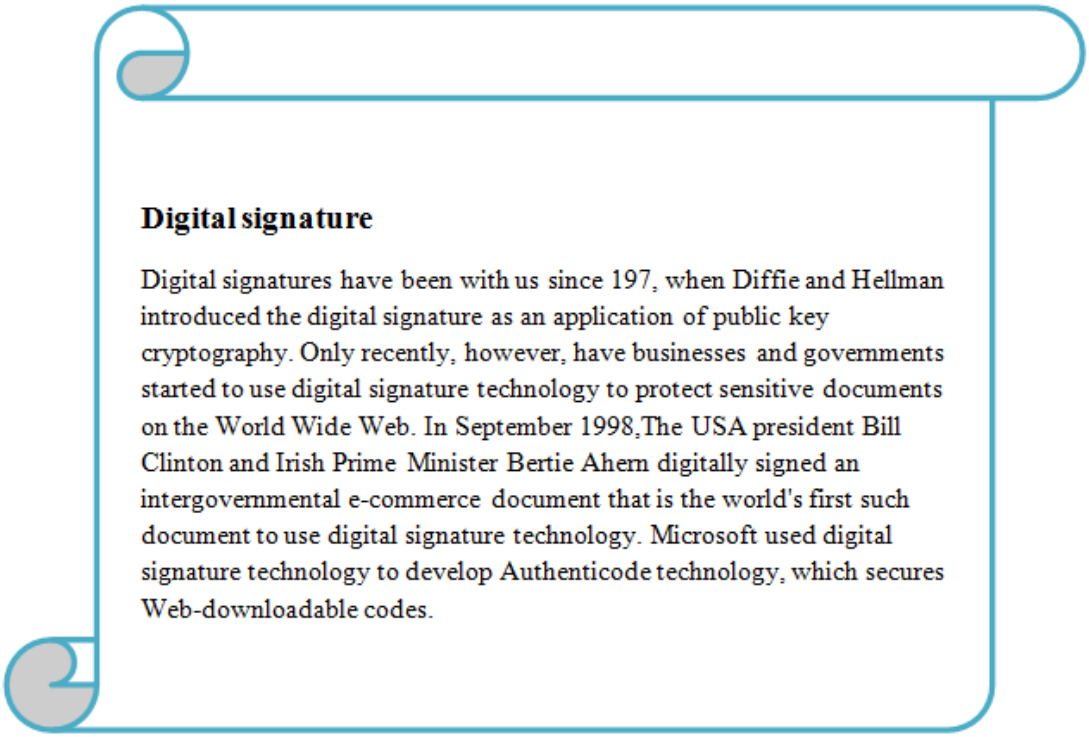
In the pre-encryption step, we obtain:
Vec=[A2 3C EC F0 27 48 D2 71 34 B5 78 39 19 15 A3 77 27 11 73 46].
And Signvector1 = [442 060 516 520 039 072 490 113 052 461 120 057 025 021 443 19],
In the Sign computation step, we obtain the following:
Signvector2 = [058 031 474 121 006 000 444 490 430 445 057 025 021 443 119].
The generated signature is appended to the text file to obtain the signed message.

Experiment 2

In this experiment, we verify the message by the public key of the sender (e). We find that the hash code of the received message and the hash code of the signature both are equal: [A2 3C EC F0 27 48 D2 71 34 B5 78 39 19 15 A3 77 27 11 73 46]. So, the signature is correct and the message is authenticated and not altered during the transmission. If we use another public key such as e=36. Then, we obtain the following hash code: [B2 3D AC F0 22 33 A2 71 34 B5 12 39 69 55 33 77 27 11 73 11]. Which is very different from the original hash code. So, the received message is not authorized and the signature is incorrect.

Experiment 3

In this experiment, we test the sensitivity of the hash function algorithms SHA- 1 and MD5 by altering only 1-byte of the message during the transmission. We compare the hash code of our text and the hash code of the following text (only remove'6' from the first line):



Digital signature

Digital signatures have been with us since 197, when Diffie and Hellman introduced the digital signature as an application of public key cryptography. Only recently, however, have businesses and governments started to use digital signature technology to protect sensitive documents on the World Wide Web. In September 1998,The USA president Bill Clinton and Irish Prime Minister Bertie Ahern digitally signed an intergovernmental e-commerce document that is the world's first such document to use digital signature technology. Microsoft used digital signature technology to develop Authenticode technology, which secures Web-downloadable codes.

We obtain the following hash codes results:

Hash code result from SHA-1 algorithm =(65 53 5B 7A 1C F9 CF 94 82 28 74 19 02 5C D0 ED 3B E4 EC E4) $_{Hex}$.

Hash code result from MD5 algorithm =(6E 77 96 49 2A E6 94 1F 73 9F 9E 37 8B 87 A5 0E) $_{Hex}$.

In addition any altered or change in message leads to change in sending time that is mean the message was forged.

4. Conclusions

Our proposed algorithm for digital signature provided a high level of reliability and security of data transmitted within the network computers.
The presented digital signature algorithm has the following properties:
1. Only the sender can sign his document (with his private key). Everyone can verify the signature (with the sender public key).
2. The recipient is sure that the sender signed the message.
3. Message cannot be altered since that would be detected through verification.
4. The recipient of the message can prove that the sender had actually send the message.

5. References

[1] William Stalling, "Cryptography and Network Security, Principles and Practice", Third Edition, Prentice Hall, 2003.
[2] R. Rivest, Shamir A. and Adleman L., ”A Method for Obtaining Digital Signatures and Public Key ***Cryptosystems”, Commun. ACM, Vol 21 , pp. 120-126, April 1978.***
[3] Aiad Ibraheem Abdul-Sada, "Text File Digital Signature", Journal of AL-Qadisiya for Computer Science and Mathematics, Vol. 2, No. 2, pp. 128-139, Qadisiya, Iraq, 2010.
[4] R. Rivest , "The MD5 Message-Digest Algorithm", MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992
[5] Chen Tianhuang, Xu Xiaoguang, " Digital Signature in The Application of E-Commerce Security", 2010 International Conference on E-Health Networking, Digital Ecosystems and Technologies, China, 2010.