ISSN No. 0976-5697

Volume 3, No. 2, March-April 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Modified MD5 Algorithm for Wireless Networks

Dr. A.K. Santra
Professor and Dean,
MCA Department, CARE School of Computer Applications,
Tiruchirappalli, Tamil Nadu.
dr.c.n.srinivasan@gmail.com

Nagarajan S *
Research Scholar,
Bharathiar University, Coimbatore and Professor and Head,
The Oxford College of Science,
Bangalore, Karnataka.

Abstract: The application of technological and related procedures to safeguard the security of various documents while moving on the channel is an important responsibility in electronic data systems. This paper specifies the modification of the MD5 which may organizations to protect sensitive data. Protection of data during transmission or while in storage may be necessary to maintain the confidentiality and integrity of the information represented by the data. The algorithms uniquely define the mathematical steps required to transform data into a cryptographic cipher and also to transform the cipher back to the original form. The Data Encryption Standard is being made available for use by various agencies within the context of a total security consisting of physical security procedures, good information management practices, and computer system/network access controls.

Key words: computer security, data encryption standard, triple data encryption algorithm, Federal

I. MD5 ALGORITHM

A. MD5 Algorithm Description:

MD5 algorithm begin by supposing that it have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. Imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1}

The following five steps are performed to compute the message digest of the message.

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.

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.

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))

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, the 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. 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. The elements of the table are given in the appendix. 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] [BCDA3224]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA7228]
[ABCD879][DABC91210][CDAB101711] [BCDA1122 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. */
[ABCD1517] [DABC6918] [CDAB111419][BCDA0 20 20]
[ABCD5521][DABC10922][CDAB151423][BCDA420 24]
[ABCD9525][DABC14926][CDAB31427][BCDA 8 20 28]
[ABCD13529][DABC2930][CDAB71431][BCDA122032]
/* 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. */
[ABCD5433][DABC81134][CDAB111635][BCDA142336]
[ABCD1437][DABC41138][CDAB71639][BCDA102340]
[ABCD13441][DABC01142][CDAB31643][BCDA62344]
[ABCD9445][DABC121146][CDAB151647][BCDA22348]
/* 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. */
[ABCD0649][DABC71050][CDAB141551][BCDA52152]
[ABCD12653][DABC31054][CDAB101555][BCDA12156]
[ABCD8657][DABC151058][CDAB61559][BCDA132160]
[ABCD4661][DABC111062][CDAB21563][BCDA92164]
/* 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, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

II. PROPOSED MODIFICATION ON DES AND TRIPPLE DES ALGORITHM

The initial 64 bit key to be generated is generated using the RSA algorithm. The procedure is described in the following steps:

Step1: The 256 bit decimal output generated from the RSA is taken as the initial value to start with.

Step 2: Using random generator algorithm, randomly 64 bits are extracted from the 256 bit decimal output of the RSA which becomes the initial key for the DES and triple DES algorithms.

The proposed random generator algorithm is as shown in the flow chart below.

III. PROOF OF RESULT

Theorem 1 (Fermat's Little Theorem) *If* p *is a prime number, and* a *is an integer such that* (a, p) = 1, *then* ap−1 = 1(mod p).

Proof: Consider the numbers (a · 1), (a · 2), . . . (a · (p − 1)), all modulo p. They are all different. If any of them were the same, say a · m = a · n(mod p), then a · (m − n) = 0(mod p) so m− n must be a multiple of p. But since all m and n are less than p, m = n. Thus a·1, a·2, . . . , a·−(p1) must be a rearrangement of 1, 2, . . . , (p −1). So modulo p, we have: $\prod_{i=1}^{p-1} i = \prod_{i=1}^{p-1} a.i = a^{p-1}\prod_{i=1}^{p-1} i$
so a^{p-1} = 1(mod p).

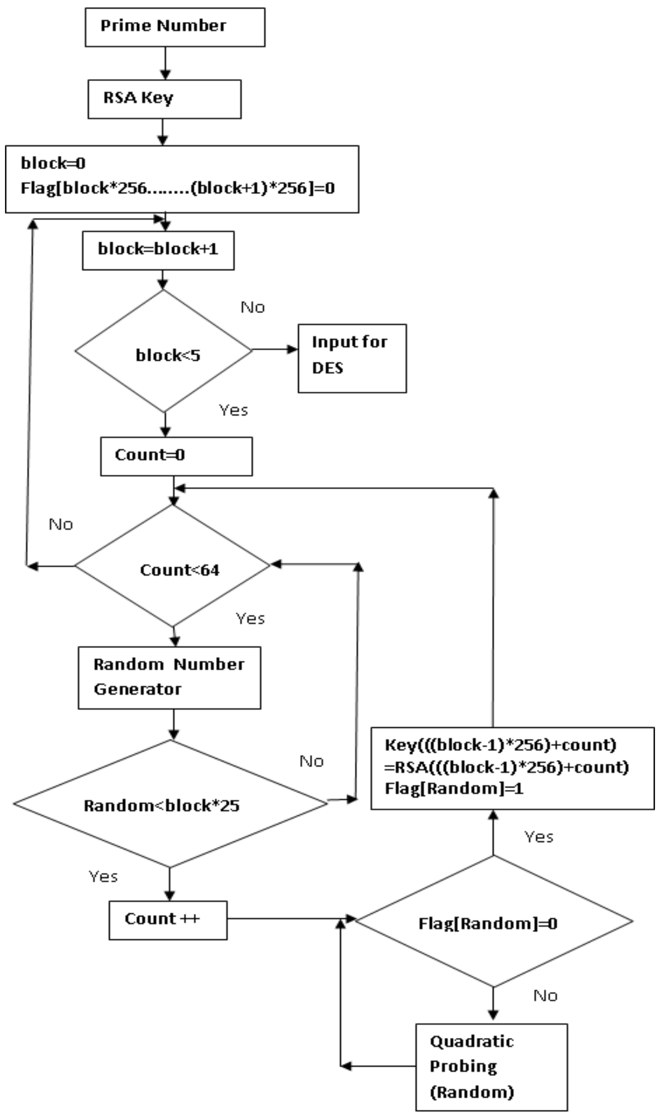

Figure.1 Proposed DES algorithm

Theorem 2 (Fermat's Theorem Extension) If (a,m) = 1 then $a^{\Phi(m)}$ = 1(mod m),where Φ(m) is the number of integers less than m that are relatively prime to m. The number m is not necessarily prime.

Proof: Same idea as above. Suppose Φ (m) = n. Then suppose that the n numbers less than m that are relatively prime to m are: a1, a2, a3, . . . , an. Then a · a1, a · a2, . . . , a an are also relatively prime to m, and must all be different, so they must just be a rearrangement of the a1, . . . , an in some order. Thus: $\prod_{i=1}^{n} ai = \prod_{i=1}^{n} a.a.i = a^{n}\prod_{i=1}^{n} ai$
modulo m, so a^{n} = 1(mod m).

***Theorem 3 (Chinese Remainder Theorem)** Let* p *and* q *be two numbers (not necessarily primes), but such that* (p, q) = 1. *Then if* a = b(mod p) *and* a = b(mod q) *we have* a = b(mod pq).

Proof: If a = b(mod p) then p divides (a− b). Similarly, q divides (a− b). But p and q are relatively prime, so pq divides (a − b). Consequently, a = b(mod pq). (This is a special case with only two factors of what is usually called the Chinese remainder theorem .)

A. *Proof of the Main Result:*

Based on the theorems above, here is why the RSA encryption scheme works. Let p and q be two different (large) prime numbers, let $0 \leq M < pq$ be a secret message1, let d be an integer (usually small) that is relatively prime to $(p - 1)(q - 1)$, and let e be a number such that de = 1(mod (p − 1)(q − 1)). The encoded message is $C = M^e$(mod pq), so we need to show that the decoded message is given byM = C^d(mod pq).

Proof: Since de = 1(mod (p−1)(q −1)), de = 1+k(p−1)(q −1) for some integer k. Thus: Cd = Mde = M1+k(p−1)(q−1) = M · (M(p-1)(q−1))k. If M is relatively prime to p, then $M^{de} = M \cdot (M^{p-1})^{k(q-1)} = M \cdot (1)^{k(q-1)} = M$(mod p)

By the extension of Fermat's Theorem giving M^{p-1} = 1(mod p) followed by a multiplication of both sides byM. But ifM is not relatively prime to p, thenM is a multiple of p, so equation 1 still holds because both sides will be zero, modulo p. By exactly the same reasoning, $M^{de} = M \cdot M^{q-1}$ = M(mod q). If we apply the Chinese remainder theorem to equations 1 and 2, we obtain the result we want: M^{de} = M(mod pq). Finally, given the integer d, we will need to be able to find another integer e such that de=1(mod(p−1)(q−1)). To do so we can use the extension of Fermat's theorem to get $d^{\Phi((p-1)(q-1))}$ = 1(mod (p−1)(q−1)), so $d^{\Phi((p-1)(q-1))-1}$(mod (p−1)(q−1)) is a suitable value for e.

IV. MATHEMATICAL PROOF OF THE RSA

8.1 Algorithm Key generation for RSA public-key encryption

Each entity creates an RSA public key and a corresponding private key. Each entity *A* should do the following:

a) Generate two large random (and distinct) primes *p* and *q*, each roughly the same size.
b) Compute *n = pq* and *Φ = (p — l)(q —* 1). (See Note 8.5.)
c) Select a random integer e, 1 < e < *Φ*, such that gcd(e, *Φ*) = 1.
d) Use the extended Euclidean algorithm to compute the unique integer *d,1 < d < Φ*, such that *ed* = 1 (mod *Φ).*
e) *A's* public key is (n, e); *A's* private key is *d.*

The integers e and *d* in RSA key generation are called the *encryption exponent* and the *decryption exponent,* respectively, while *n* is called the *modulus.*

A. *RSA public-key encryption:*

B encrypts a message *m* for *A*, which *A* decrypts.

a. *Encryption. B* should do the following:

a) Obtain A's authentic public key (n, e).
b) Represent the message as an integer *m* in the interval [0, *n*—1].
c) Compute c = m^e mod *n.*
d) Send the ciphertext c to *A.*

b. *Decryption.* To recover plaintext *m* from c, *A* should do the following:

i. Use the private key *d* to recover $m = c^d$ mod *n. Proof that decryption works.* Since *ed* = 1 (mod *Φ),* there exists an integer *k* such that *ed* = 1 + k *Φ*. Now, if *gcd(m,p)* = 1 then by Fermat's theorem m^{p-1} = 1 (mod *p).* Raising both sides of this congruence to the power *k(q* — 1) and then multiplying both sides by *m* yields ml+k(p-l)(q-l) = *m* (mod *p)*

On the other hand, if gcd (m, p) = *p*, then this last congruence is again valid since each side is congruent to 0 modulo *p.* Hence, in all cases $m^{ed} = m$ (mod *p).*By the same argument, m^{ed} =m(mod *q).* Finally, since *p* and *q* are distinct primes, it follows that $m^{ed} = m$ (mod *n),* and, hence, $c^d = (m^e)^d = m$ (mod *n).*

a. *(RSA encryption with artificially small parameters)*

Key generation. Entity *A* chooses the primes *p* = 2357, *q* = 2551, and computes *n = pq* = 6012707 and *Φ = (p* —1)(q ‾ 1) = 6007800. *A* chooses e = 3674911 and, using the extended Euclidean algorithm, finds *d* = 422191 such that *ed* = 1 (mod *<p). A's* public key is the pair (n = 6012707, e = 3674911), while A's private key is *d* = 422191. *Encryption.* To encrypt a message *m* = 5234673, *B* uses an algorithm for modular exponentiation (e.g., Algorithm 2.143) to compute c= m^e mod *n* = $5234673^{3674911}$ mod 6012707 = 3650502, and sends this to *A. Decryption.* To decrypt c, *A* computes c^d mod *n* = 3650502^{422191} mod 6012707 = 5234673. *(universal exponent)* The number λ = lcm(p — *l,q*— 1), sometimes called the *universal exponent* of *n,* may be used instead of *Φ=(p*—1)*(q*—1) in RSA key generation. Observe that λ is a proper divisor of *Φ*. Using λ can result in a smaller decryption exponent *d,* which may result in faster decryption. However, if *p* and *q* are chosen at random, then *gcd(p* —1, *q*— 1) is expected to be small, and consequently *Φ* and λ will be roughly of the same size.

B. *4.2 Security of RSA:*

This subsection discusses various security issues related to RSA encryption. Various attacks which have been studied in the literature are presented, as well as appropriate measures to counteract these threats.

a. *Relation to factoring:*

The task faced by a passive adversary is that of recovering plaintext *m* from the corresponding ciphertext c, given the public information (n, e) of the intended receiver *A.* This is called the *RSA problem* (RSAP).There is no efficient algorithm known for this problem.

One possible approach which an adversary could employ to solving the RSA problem is to first factor *n,* and then compute *Φ* and *d* . Once *d* is obtained, the adversary can decrypt any ciphertext intended for *A.*

On the other hand, if an adversary could somehow compute *d,* then it could subsequently factor *n* efficiently as follows. First note that since *ed* = 1 (mod *Φ),* there is an integer *k* such that *ed* — 1 = *k Φ*. Hence, a^{eti_1} = 1 (mod *n)* for all *a* Є Z*. Let *ed* — 1 = $2^s t$, where *t* is an odd integer. Then it can be shown that there exists an *i* Є [l,s] such that $a^{2i-1t} \neq \pm 1$ (mod *n)* anda^{2it} = 1 (mod *n)* for at least half of all *a*

Є Z_n; if *a* and *i* are such integers then gcd(a—1, *n)* is a non-trivial factor of *n.* Thus the adversary simply needs to repeatedly select random *a* ЄZ* and check if an *i* Є [1, s] satisfying the above property exists; the expected number of trials before a non-trivial factor of *n* is obtained is 2. This discussion establishes the following.

The problem of computing the RSA decryption exponent *d* from the public key (n, e), and the problem of factoring *n,* are computationally equivalent. Then generating RSA keys, it is imperative that the primes *p* and *q* be selected in such a way that factoring *n = pq* is computationally infeasible.

b. Small Encryption Exponent e:

In order to improve the efficiency of encryption, it is desirable to select a small encryption exponent e such as e = 3. A group of entities may all have the same encryption exponent e, however, each entity in the group must have its own distinct modulus. If an entity *A* wishes to send the same message *m* to three entities whose public moduli are n_1, n_2, n_3 and whose encryption exponents are e = 3, then *A* would send *Ci* = m^3 mod n; for *i* = 1,2,3. Since these moduli are most likely pairwise relatively prime, an eavesdropper observing c_1, C_2,C_3 can use Gauss's algorithm to find a solution *x*, 0≤ *x* < $n_1n_2n_3$, to the three congruences *x* = c_1 (mod n_1) *x* =C2(mod n_2) *x*=C3(mod n_3). Since $m^3 < n_1n_2n_3$, by the Chinese remainder theorem, it must be the case that x = m^3. Hence, by computing the integer cube root of x, the eavesdropper can recover the plaintext m. Thus a small encryption exponent such as e = 3 should not be used if the same message, or even the same message with known variations, is sent to many entities. Alternatively, to prevent against such an attack, a pseudorandomly generated bitstring of appropriate length should be appended to the plaintext message prior to encryption; the pseudorandom bit-string should be independently generated for each encryption. This process is sometimes referred to as *salting* the message.

Small encryption exponents are also a problem for small messages m, because if m< $n^1/^e$, then m can be recovered from the ciphertext c = m^e mod *n* simply by computing the integer e^{th} root of c; salting plaintext messages also circumvents this problem.

c. Forward Search Attack:

If the message space is small or predictable, an adversary can decrypt a ciphertext c by simply encrypting all possible plaintext messages until c is obtained. Salting the message as described above is one simple method of preventing such an attack.

d. Small Decryption Exponent d:

As was the case with the encryption exponent e, it may seem desirable to select a small decryption exponent *d* in order to improve the efficiency of decryption.^{x} However, if gcd *(p*—1, *q*—1) is small, as is typically the case, and if *d* has up to approximately one-quarter as many bits as the modulus *n,* then there is an efficient algorithm (referenced on page 313) for computing *d* from the public information (n, e). This algorithm cannot be extended to the case where *d* is approximately the same size as *n.* Hence, to avoid this attack, the decryption exponent *d* should be roughly the same size as *n.*

e. Multiplicative Properties:

Let m_1 and m_2 be two plaintext messages, and let C_1 and C_2 be their respective RSA encryptions. Observe that $(m_1m_2)^e$ = $m_1^e m_2^e$ = C1C2 (mod *n).*In other words, the ciphertext corresponding to the plaintext *m* = m_1m_2 mod *n* is c = c_1c_2 mod n; this is sometimes referred to as the *homomorphic property* of RSA. This observation leads to the following *adaptive chosen-ciphertext attack* on RSA encryption.

Suppose that an active adversary wishes to decrypt a particular ciphertext c = m^e mod *n* intended for *A.* Suppose also that *A* will decrypt arbitrary ciphertext for the adversary, other than c itself. The adversary can conceal c by selecting a random integer *x* Є Z_n* and computing c^- = cx^e mod *n.* Upon presentation of c^-, *A* will compute for the adversary m^- = $(c^-)^d$ mod n. Since m^- = $(c^-)^d$ = c^d $(x^e)^d$ = mx (mod n), the adversary can then compute *m* =m^-x^{-1} mod *n.*

This adaptive chosen-ciphertext attack should be circumvented in practice by imposing some structural constraints onplaintext messages. If a ciphertext c is decrypted to a message not possessing this structure, then c is rejected by the decryptor as being fraudulent. Now, if a plaintext message *m* has this (carefully chosen) structure, then with high probability *mx* mod *n* will not for *x*Є Z_n*. Thus the adaptive chosen-ciphertext attack described in the previous paragraph will fail because *A* will not decrypt c for the adversary.

f. Common Modulus Attack:

The following discussion demonstrates why it is imperative for each entity to choose its own RSA modulus *n.* It is sometimes suggested that a central trusted authority should select a single RSA modulus *n,* and then distribute a distinct encryption/decryption exponent pair (e_i, d_i) to each entity in a network. However, as shown in (i) above, knowledge of any (e_i, d_i) pair allows for the factorization of the modulus *n,* and hence any entity could subsequently determine the decryption exponents of all other entities in the network. Also, if a single message were encrypted and sent to two or more entities in the network, then there is a technique by which an eavesdropper (any entity not in the network) could recover the message with high probability using only publicly available information.

g. Cycling Attac#ks:

Let *c* = m^e mod *n* be a ciphertext. Let *k* be a positive integer such that c^{ek}=*c*(mod *n);* since encryption is a permutation on the message space {0,1,... , n — 1} such an integer *k* must exist. For the same reason it must be the case that c^{k-1} = *m* (mod *n).* This observation leads to the following *cycling attack* on RSA encryption. An adversary computes c^e mod *n,* c^{e2} mod *n,* c^{e3} mod *n*,... until c is obtained for the first time. If c^{ek} mod *n* =c, then the previous number in the cycle, namely c $^{ek-1}$ mod *n,* is equal to the plaintext *m. A generalized cycling attack* is to find the smallest positive integer *u* such that f =gcd(c^e—*c,n)* > 1. If c^e=*c* (mod *p)* and $c^e \neq c$ (mod *q)* then *f* = *p.* Similarly, if $c^e \neq c$ (mod *p)* and c^e = *c* (mod *q* (8.2) then 1 = *q.* In either case, *n* has been factored, and the adversary can recover *d* and then *m.* On the other hand, if both *c* =*c* (mod *p)* and *c* =*c* (mod *q),*(8.3) then *f* = *n* and c^e = *c* (mod *n).* In fact, *u* must be the smallest positive integer *k* for which c^e = *c* (mod *n).* In this case, the basic cycling attack has succeeded and so 7n = c mod n can be computed efficiently. The generalized cycling attack usually terminates before the cycling attack does. For this reason, the generalized cycling attack can be viewed as being essentially an algorithm for factoring *n*. Since factoring *n* is assumed to be intractable, these cycling attacks do not pose a threat to the security of RSA encryption.

h. Message Concealing:

A plaintext message *m*, 0 < m < n—1, in the RSA public-key encryption scheme is said to be *unconcealed* if it encrypts to itself; that is, $m^e = m$ (mod *n)*. There are always some messages which are unconcealed (for example *m* = 0, *m* = 1, and *m* = *n*—1). In fact, the number of unconcealed messages is exactly [1 + gcd(e—*l* , *p* —1)] • [1 + gcd(e—1,*q*—1)]. Since *e — l,p—l* and *q — 1* are all even, the number of unconcealed messages is always at least 9. If *p* and *q* are random primes, and if e is chosen at random (or if e is chosen to be a small number such as e = 3or e = 2^{16} + l = 65537), then the proportion of messages which are unconcealed by RSA encryption will, in general, be negligibly small, and hence unconcealed messages do not pose a threat to the security of RSA encryption in practice.

C. RSA Encryption in Practice:

There are numerous ways of speeding up RSA encryption and decryption in software and hardware implementations. Some of these techniques are covered in Chapter 14, including fast modular multiplication, fast modular exponentiation, and the use of the Chinese remainder theorem for faster decryption. Even with these improvements, RSA encryption/decryption is substantially slower than the commonly used symmetric-key encryption algorithms such as DES. In practice, RSA encryption is most commonly used for the transport of symmetric-key encryption algorithm keys and for the encryption of small data items.

The RSA cryptosystem has been patented in the U.S. and Canada. Several standards organizations have written, or are in the process of writing, standards that address the use of the RSA cryptosystem for encryption, digital signatures, and key establishment. For discussion of patent and standards issues related to RSA.

(recommended size of modulus) Given the latest progress in algorithms for factoring integers , a 512-bit modulus *n* provides only marginal security from concerted attack. As of 1996, in order to foil the powerful quadratic sieve and number field sieve factoring algorithms, a modulus *n* of at least 768 bits is recommended. For long-term security, 1024-bit or larger moduli should be used.

a. (selecting primes):

a) The primes *p* and *q* should be selected so that factoring *n* = *pq* is computationally infeasible. The major restriction *onp* and *q* in order to avoid the elliptic curve factoring algorithm is that *p* and *q* should be about the same bitlength, and sufficiently large. For example, if a 1024-bit modulus *n* is to be used, then each of *p* and *q* should be about 512 bits in lengt.
b) Another restriction on the primes *p* and *q* is that the difference *p*—*q* should not be too small. If *p*—*q* is small, then *p* ≈*q* and hence *p* ≈√*n*. Thus, *n* could be factored efficiently simply by trial division by all odd integers close to √*n. If p* and *q* are chosen at random, then *p*—*q* will be appropriately large with overwhelming probability.
c) In addition to these restrictions, many authors have recommended that *p* and *q* be strong primes. A prime *p* is said to be a *strong prime* if the following three conditions are satisfied:
   a) *p*—1 has a large prime factor, denoted r;
   b) *p* + 1 has a large prime factor; and
   c) *r*—1 has a large prime factor.

The reason for condition (a) is to foil Pollard's *p*—1 factoring algorithm which is efficient only if *n* has a prime factor *p* such that *p*—1 is smooth. Condition (b) foils the *p* + 1 factoring algorithm mentioned, which is efficient only if *n* has a prime factor *p* such that *p* + 1 is smooth. Finally, condition (c) ensures that the cycling attacks will fail. If the prime *p* is randomly chosen and is sufficiently large, then both *p—1* and *p+1* can be expected to have large prime factors. In any case, while strong primes protect against the *p—1* and *p+1* factoring algorithms, they do not protect against their generalization. The latter is successful in factoring *n* if a randomly chosen number of the same size as *p* has only small prime factors. Additionally, it has been shown that the chances of a cycling attack succeeding are negligible if p and *q* are randomly chosen. Thus, strong primes offer little protection beyond that offered by random primes. Given the current state of knowledge of factoring algorithms, there is no compelling reason for requiring the use of strong primes in RSA key generation. On the other hand, they are no less secure than random primes, and require only minimal additional running time to compute; thus there is little real additional cost in using them.

b. (small encryption exponents):

If the encryption exponent e is chosen at random, then RSA encryption using the repeated square-and-multiply algorithm takes *k* modular squarings and an expected *k/2* (less with optimizations) modular multiplications, where *k* is the bitlength of the modulus *n*. Encryption can be sped up by selecting e to be small and/or by selecting e with a small number of 1 's in its binary representation. The encryption exponent e = 3 is commonly used in practice; in this case, it is necessary that neither *p*—1 nor *q*—1 be divisible by 3. This results in a very fast encryption operation since encryption only requires 1 modular multiplication and 1 modular squaring. Another encryption exponent used in practice is e = 2^{16} + 1 = 65537. This number has only two 1's in its binary representation, and so encryption using the repeated square-and-multiply algorithm requires only 16 modular squarings and 1 modular multiplication. The encryption exponent e = 2^{16} + 1 has the advantage over e = 3 in that it resists the kind of attack , since it is unlikely the same message will be sent to 2^{16}+1 recipients.

V. CONCLUSION

In this paper the proposal is to modify the MD5 algorithm to improve the hashing information exchanged between any two nodes on the network. In its present form it can be broken. By the proposed modification the purpose is to enhance the time to break so that with the timestamp for the transfer of the frame the information would have already reached the destination and action accordingly taken as needed. This enhances the performance of the MD5 algorithm to a large extent. It is very clear with the proof given above. For future research on this, the inclusion of the knowledge of some of the other theorems of number theory can be use to further enhance the performance of the MD5 algorithm.

VI. REFERENCES

[1]. "Cryptographic Algorithms for Protection of Computer Data During Transmission and Dormant Storage," Federal Register **38**, No. 93 (May 15, 1973).

[2]. Data Encryption Standard, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (January 1977).

[3]. Carl H. Meyer and Stephen M. Matyas, Cryptography: A New Dimension in Computer Data Security, John Wiley & Sons, New York, 1982.

[4]. Dorthy Elizabeth Robling Denning, Cryptography and Data Security, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

[5]. D.W. Davies and W.L. Price, Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronics Funds Transfer, Second Edition, John Wiley & Sons, New York, 1984, 1989.

[6]. Miles E. Smid and Dennis K. Branstad, "The Data Encryption Standard: Past and Future," in Gustavus J. Simmons, ed., Contemporary Cryptography: The Science of Information Integrity, IEEE Press, 1992.

[7]. Douglas R. Stinson, Cryptography: Theory and Practice, CRC Press, Boca Raton, 1995.

[8]. Bruce Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, New York, 1996.

[9]. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, 1997.

10.