Development of Cryptographic Algorithm for Secure Communication

[1]Sahnoaj Ahmed, [2]Bijoy Mandal, [3]Dr. Arindam Biswas, [4]A. K. Bhattacharjee
[1]Dumkal Institute of Engineering & Technology, Dumkal, Dumkal, WB, India
[2,3]Dept. of CSE and ECE, NFET, NSHM Knowledge Campus, Durgapur, WB, India
[4]Dept.of Electronics and Communication Engineering, N. I. T., Durgapur, India

Abstract
RSA algorithm uses Integer Factorization as the backbone of data security while Diffie-Hellman Algorithm uses Discrete Logarithm to provide platform for secure key exchange. In this paper, we provide algorithm to merge both of them to provide a user with even higher level of data security. Actually, our intent is to secure data of smaller as well as larger size by obtaining one randomly chosen key pair from set of RSA keys and one randomly chosen secret key using Diffie-Hellman algorithm and then applying RSA encryption to make even public components of Diffie-Hellman algorithm inaccessible for any eavesdropper freely. We design the encryption method, which uses Diffie-Hellman Secret Key in such a way that it would not only encrypt the data but also does not increase the size of it any further. On top of that, encrypted exchange of public components of Diffie-Hellman part of the system makes it hidden from all other than intended users to see Diffie-Hellman generator and prime modulus. This will make our algorithm M*N times complex to break using even the latest version of Brute Force attack, where M and N are corresponding complexities imposed by the Diffie-Hellman and RSA algorithms respectively.

Keywords
Cryptography, RSA Diffie-Hellman, Cipher Systems, Hybrid Cryptographic Schemes

I. Introduction

In traditional cryptography, cipher systems have been analyzed by modeling cryptographic algorithms as ideal mathematical objects. If any attacker is able to find any sophisticated strategy, this would lead to a low-complexity algorithm for the solution of a problem commonly believed to be intractable. This can be shown by complexity-theoretical reductions. Trust gained in this way is independent of the concrete implementation of a cryptographic algorithm. RSA algorithm for Integer Factorization [1-3] and Diffie-Hellman Algorithm for Discrete Logarithm [4-5] is already exists in literature to provide a platform for secure key exchange [6-8]. RSA encryption is secured because it imposes immense computational complexity to a "Hacker" by forcing him to go through billions and even more of computations in order to find an exact "Private Key" [9]. However, in the process it appears that some large volume of data becomes almost impossible to handle under RSA encryption technique. Besides security of RSA, encryption also depends on length of the keys used. With larger keys, encryption becomes more complex to the end users in the time domain. On the other hand, encrypted exchange of public components of Diffie-Hellman part of the system makes it hidden from all other than intended users to see Diffie-Hellman generator and prime modulus in already exists in literature [10]. As it is a well-known fact that RSA algorithm is used in PKC (Public Key Cryptosystem) [11-12] because of its unique nature of providing means for User Authentication [13-14] and Diffie-Hellman algorithm for its simple yet very effective Secure Key Exchange property. Hence, we shall use both of them to good effect.
There are two publicly disclosed prime numbers known as generator (g) and modulus (n) are used in Diffie-Hellman key Exchange algorithm. If these two are exchanged among parties using RSA encryption, then to find such Diffie-Hellman Secret Key one has to break the RSA encryption. Thus, it will be immensely difficult for an eavesdropper to find the secret key, which can then be used by end users for encryption and decryption purposes. Of course, RSA will be employed for the User Authentication purpose. In this fashion the computational complexity does not increase for end users and at the same time data is also M*N times more secure; M, N are the complexities of solving DLP(Discrete Logarithm Problem) and RSA Problem [15] respectively.

II. Points of Concern Related to Cryptographic Algorithm

Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet. There are several ways of classifying cryptographic algorithms. For purposes of this paper, they will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use.

A. Secure Key Exchange

Generally in Hybrid Cryptographic schemes [16-17] Session Key [18-19] is used as secret key for encryption decryption purposes following the mechanism of Secret Key Cryptography and the secret session key itself is exchanged between communicating parties using Public Key Encryption Decryption policy. In our own approach we do not wish to send encrypted secret key along with encrypted data rather we try to generate same secret key at both ends using Diffie-Hellman key exchange policy.
More over we exchange public components of this algorithm keeping them encrypted with RSA encryption decryption technique and therefore Secret Keys are far from being hacked.

B. User Authentication

RSA algorithm is used in this system to provide platform for authentication for users connected through insecure channels. Using well-known Hash functions [20] a hash value can be generated which will be encrypted with senders' private key and then can be sent as a digital signature along with Diffie-Hellman public components.

It is important that while exchanging Diffie-Hellman public components a Hash function is used to compute hash value for users' authentication. Because otherwise Man in the Middle [21-22] may still be left with a little chance to reveal Diffie-Hellman public components even though we kept it encrypted with RSA public key.

C. Degree of Security

The Secret keys in our system are almost impossible to be hacked because

- Even before Diffie-Hellman, public components are exchanged among parties already RSA public key exchange has taken place and Diffie-Hellman public components will be exchanged using those RSA public keys. Therefore, only intended users are able to see those public components.
- While exchanging Diffie-Hellman public components RSA is also used for users' authentication. Hence, all Man in the Middle attack can be brought to justice using Digital Signatures [9] as evidence.
- Using Brute Force attack, if somebody tries to reveal the secret key in computationally feasible time then he/she has to not only solve Integer Factorization problem in feasible time but also has to solve Discrete Logarithm problem at the same time.

D. Computational Complexity

Simply if we look at the computational complexities of both of RSA and Diffie-Hellman algorithm then we must realize that it takes more time with keys of larger size than that of smaller ones.

The fastest classical algorithm to solve Discrete Logarithm problem is the "Number Field Sieve" with heuristic time complexity [3].

$$2^{O\left(n^{1/3}\log n^{2/3}\right)}$$

On the other hand, the best-known algorithm for Integer Factorization is "GNFS" (General Number Field Sieve) which has a time complexity of Θ [3].

$$\left(\exp\left(\left(\frac{64}{9}n\right)^{1/3}(\log n)^{2/3}\right)\right)$$

Our algorithm will impose a computational complexity, which is equal to the multiplication of these two above-mentioned expressions because the public components of Diffie-Hellman are protected by RSA public key encryption.

III. Results and Discussion

A. Analysis of Diffie-Hellman algorithm:

By using Diffie-Hellman algorithm, we can find secret key, which can be employed in Public Key Cryptography, in the way given below –

A small Example:

Person A

Choose, x=119

Send to Person B: X = 113^{119} mod 107 = 96

$K_A = 74^{119}$ mod 107 = 73

Person B

Choose y=117

Send to Person A: Y = 113^{117} mod 107 = 74

$K_B = 96^{117}$ mod 107 = 73

Here both Person A and Person B select the base g (113) and the prime number n (107) through mutual consent. Now, based on their own private key and the public key learned from the other party, Person A and Person B have computed their secret keys, KA and KB, respectively, which are equal to gxy mod n.

We can see very clearly that while Person A and B are communicating with each other, as their private keys are kept secret, the Public keys KA and KB (both of which are same and value 73 in this case) are safe from any third party who might be eavesdropping.

Important Analysis: In the Secret Key Exchange technique, Secret Key generated is safe because there is no known polynomial time algorithm to solve Discrete Logarithm problem. However, a simple encryption based on this Secret Key may disclose the drawbacks of this algorithm a little further.

Suppose person A wants to send a secret password "pgeftt123" to person B.

Using his/her own secret key A encrypts the password "pgeftt123" as "*j45 Q%9" (supposedly), by varying ASCII values of the characters of the password string with the secret key value directly.

Now some third party, can see that every time there is a repetition in the input password String like "pgeftt123123" there will be some encrypted vale which is equal to "*j45 Q%9Q%9".

Therefore, it is very easy for some well-equipped eavesdropper to understand that the ASCII values of the characters of the real message is changed using the secret key value in a one to one mapping from one character set to the same character set.

This is why Secret Key Encryption algorithms like Cipher Feedback (CFB) [23] and Cipher Block Chaining (CBC) [24], use feedbacks from previous ciphers to keep repeatedly occurring Plain texts not result in same ciphers.

B. Analysis of RSA Algorithm:

General procedure of the implementation of RSA secret key cryptography With an Example:

1. Select p=3 and q=5.
2. The modulus n = pq = 15.
3. The value e must be relatively prime to (p-1) (q-1) = (2) (4) = 8.
4. Select e=11
5. The value d must be chosen so that (ed-1)/[(p-1)(q-1)] is an integer. Thus, the value (11d-1)/ [(2) (4)] = (11d-1)/8 must be an integer. Calculate one possible value, d=3.
6. Suppose we wish to send a string "Encryption", we will convert the string to the decimal representation of the BYTE values of the characters, which would be 32787959508959456666 8142.
7. The sender encrypts each digit one at a time (we have to because the modulus is so small) using the public key value (e, n) = (11, 15). Thus, each Plain text digits is reformed using the function Ci = Mi11 mod 15. Then input digit string 79595089594566668142 becomes "41770084136384147028997231958".
8. The receiver decrypts each digit using the private key value (d, n) = (3, 15). Thus, each Cipher digit is decrypted using the function Mi = Ci3 mod 15. The input digit string "41770084136384147028997231958" changes into a

digit string "79595089594566668142" and then it is again converted to plaintext string.

Now the important things that we notice are as follows:-

- In the given example above the text message is a string "Encryption"
- The Byte code "79595089594566668142" is generated from the given text. Someone may compute the ASCII code from the given text message which is "91556947314004" instead of the Byte code and then use it for further computation.
- Then this byte /ASCII code is encrypted with the RSA "Public-Key". For an example "91556947314004" is coded by Public key value (e, n) = (11, 15) as "51160146336452333".
- Now the problem is that feeding this byte code or ASCII code directly to the RSA algorithm will make this encryption easier for the "Brute-Force" attack to break it. Because in "Brute-Force" attack series of "Private-Key" s are used to decrypt the encrypted message until it finds some relevant Byte/ASCII code from which it may generate the "plain text".

Now Under Brute-Force attack someone may find the private key, which will decrypt "41770084136384147028997231958" as "79595089594566668142" and simultaneously generate the text message as "Encryption". However, if we already change the byte/ ASCII code with Secret-Key computed using the Diffie-Hellman, then even if the Brute-force attack may find the Private Key and decrypt the encrypted message, but it will not give him/her the ASCII/Byte code, rather it will give some ambiguous code for the "Code-Breaker" to understand. Therefore, he/she will not be sure if what he/she has found by decrypting the "encrypted-text" is the real byte/ASCII code or not. To make sure that Diffie-Hellman Secret key has to be found as well as the RSA private Key.

1. Analysis of Proposed Algorithm:

There are two basic steps that constitute the whole process of Data Encryption, Data Transfer, Data Decryption and those steps are **Key Exchange:** Key Exchange is done in two different parts

(i). RSA Key Exchange:

Step 1: Set Pa, Qa; both are random large prime numbers of length L bits for user A.
Step 2: Set Na = Pa * Qa (Na is called modulus) for user A.
Step 3: Select Ea that is relatively prime to (i.e., it does not divide evenly into) the product (Pa- 1)*(Qa-1). The number Ea along with Na works as the public exponent for user A.
Set Ea = generateE (Pa, Qa, l).

Step 4: Set Da = call calculateD (Pa, Qa, Ea). The Variable Da along with Na works as the secret exponent of user A.
Step 5: User A sends Na, Ea to user B.
Step 6: User B sets her own Pb, Qb; both are random large prime numbers of length L bits for user B.
User B receives Na, Ea from user A.

Step 7: Set Nb = Pb*Qb (Nb is called modulus) for user B.

Step 8: Select Eb for user B using similar way showed in step3 stated above. The number Eb along with Nb works as the public exponent for user B.
Step 9: Set Db = call calculateD (Pb, Qb, Eb). The Variable Db along with Nb works as the secret exponent of user B. User B sends Nb, Eb to user A.
Step 10: User A receives Nb, Eb.

/* All public exponents have been exchanged
Now user A posses Pa, Qa, Na, Ea, Da, Nb, Eb and user B possess Pb, Qb, Nb, Eb, Db,
Na, Ea.
Users can forget Pa, Qa and Pb, Qb */

(ii). Secret Key Exchange:

Step 1: Set p, g where p is a randomly chosen prime modulus and g is the generator for user A.
Step 2: Set x, where x is a randomly chosen large number (This is the secret of user A).
Step 3: Set Ka = g^x mod p.
Step 4: User A encrypts g, p, Ka with the public key of the intended recipient of the message and sends it to User B.
Step 5: User B receives encrypted g, p and Ka, sent by user A. User B then decrypts it and finds g, p and Ka.
Step 6: Set y, a randomly chosen large number for user B.
Step 7: Set Kb = g^y mod p. Set DHKey = (Ka) ^y mod p
Step 8: User B encrypts Kb with the public key of User A.
Step 9: User B then sends encrypted Kb to user A.
Step 10: User A receives encrypted Kb, sent by user B.
Step 11: User A decrypts it and finds Kb.
Step 12: Set DHKey = (Kb) ^x mod p (DHKey is the secret Diffie-Hellman key for user
//(DHKey is the secret Diffie-Hellman key for user B).

Using TCP connection we can send and receive data and/or keys generated through this algorithm.

As TCP is well known, it is out of the scope of this part of the discussion to elaborate on how TCP can be used to exchange Keys and/or encrypted data.

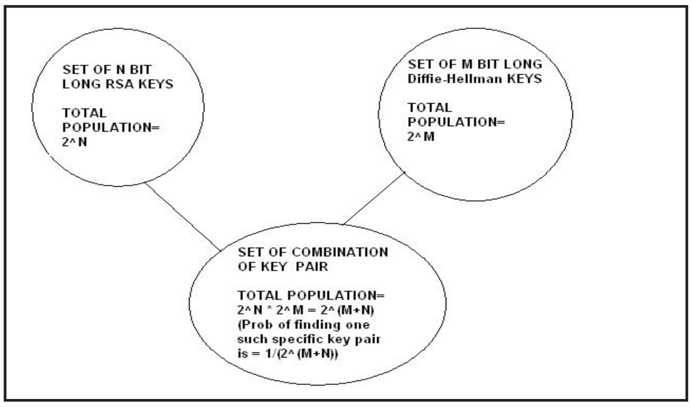
SET OF N BIT LONG RSA KEYS
TOTAL POPULATION= 2^N
SET OF M BIT LONG Diffie-Hellman KEYS
TOTAL POPULATION= 2^M
SET OF COMBINATION OF KEY PAIR
TOTAL POPULATION= 2^N * 2^M = 2^(M+N)
(Prob of finding one such specific key pair is = 1/(2^(M+N))

Fig 1: Combination of RSA and Secret Key Exchange

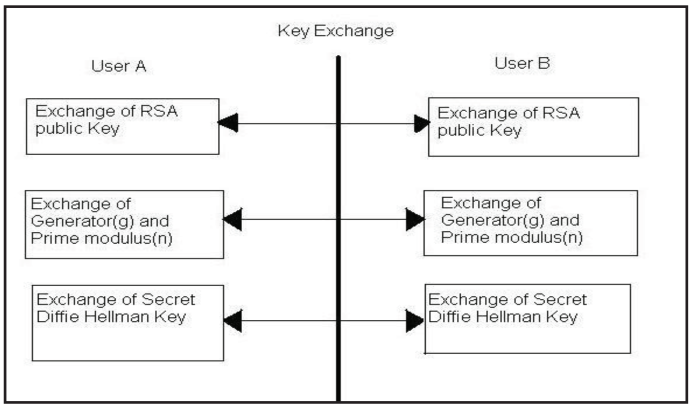
Key Exchange
User A
User B
Exchange of RSA public Key
Exchange of RSA public Key
Exchange of Generator(g) and Prime modulus(n)
Exchange of Generator(g) and Prime modulus(n)
Exchange of Secret Diffie Hellman Key
Exchange of Secret Diffie Hellman Key

Fig. 2: Key Exchange of Proposed Algorithm

2. Data Exchange

According to this algorithm, Data Exchange takes place using the following steps:

Step 1: At first user data and/or application data (assume this user as user A) is read from the system and converted into its corresponding Byte-Code.

Step 2: Now the Byte-Code is converted into its corresponding big integer form.

Step 3: Now this Byte code will be converted into Cipher using the algorithm called SecureByteConversion (Byte code) which described below.

Step 4: This Cipher is now ready to be sent form one user to the other through the vulnerable network.

Step 5: Upon receiving the Cipher, the other user will decrypt it using the algorithm called DeCipher (Cipher) which is described below.

Step 6: Finally, this Byte-Code is again transformed into the user data and/or application data and the eavesdropper is left with the Cipher-Text, which can take him computationally infeasible time to decrypt using the most efficient Brute-Force attack running on even most powerful processor/s.

C. Performance Study of Above Mentioned Algorithm:

1. Secure byte code conversion with Diffie-Hellman Key:

(i). SecureByteConversion (Byte code)

Step 1: Count the no of digits as length of the byte code of the real text message.

Step 2: a) If the length of the byte code is smaller than half the length (in digits) of the Diffie-Hellman key then go to Step 4.

b) Else if the length of the byte code is larger than the Diffie Hellman Key size then (Normally it is likely to be so, considering that the text message itself is not smaller than sixty six characters against a 512 bit Diffie-Hellman Key), then break it into segments of same size as the Diffie Hellman Key. For each such segment of the byte code perform the task defined in step 3.

c) Otherwise, go to step6.

Step 3: We keep on adding digit by digit of the byte code of the message to the corresponding digit of the Diffie-Hellman Key until all the digits of the key are spent and at the same time, we take modulo of each addition with the help of table 1 which shows the modulo10 chart for addition.

Repeat this step for all same sized segments until the final segment, which may be smaller or equal to the length (in digits) of the Diffie-Hellman key, is encountered. When it happens, go to step 5.

Step 4: Set Counter = 0.

Call rsa_encrypt (Byte coded message)

/*rsa_encrypt will return an integer which is equal to the original byte coded integer being moderated using modular residue calculation method by RSA public modulus n after it has already been raised public exponent e times to its own power.[cipher= m^e mod n] */

Check to see if the size of the integer (length in no. of digits) becomes more than half of the length of the Diffie-Hellman Key.

a) If it is so then concatenate five times more numbers of zeroes as the value of counter at the end of the code and return to Step2.

b) Increment Counter by 1 and repeat Step 4.

Step 5: Cut segment of same size as that of the byte code comprising consecutive digits from the Diffie-Hellman Key and then add it with the code using addition modulo ten arithmetic provided in Table 1.

Table 1: Addition modulo 10

(J)	(I)									
	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

X ij= I + J mod 10; where 0<= I, J<=0.
E.g., X00 = 0+0 mod 10 = 0 and X87 = 8+7 mod 10 = 5.

Step 6: Keep on taking addition modulo 10 for all digits appearing in the final segment with Diffie-Hellman key until the final digit of the byte code is dealt with.

(ii). DeCipher (Cipher)

//This algorithm decrypts the cipher and gives the byte code of the original message.

Step 1: The cipher has to be treated digit by digit and for every single digit that we encounter in the cipher, we find corresponding digit in the Diffie-Hellman Key (abbreviated as Diffie key). Now Table1's row index is searched to find the Diffie-Key digit.

Step 2: The row in which the Diffie-key is found, that particular row is further searched to find the corresponding cipher digit in it.

Step 3: Now we look up to the column index of the searched value found in previous step.

This column value is the byte code of the message.

Step 4: Now check from the end if there are five or more number of consecutive zeroes at the end of the code as delimiter.

If so, then count the number of zeroes and

Set delimiter = No of consecutive zeroes from the end of the code.

Set count = delimiter / 5.

Step 5: Call rsa_decrypt (code generated in the prev.step)

/*rsa_decrypt () gives c^d mod n where c stands for code generated in the previous step and d, n are private exponent and modulus respectively.*/

Set Count=Count -1.

Repeat Step 5 until Count=0.

Function BigInteger generate_e (BigInteger p, BigInteger q, int bitsize)

Step1: Declare Variable (BigInteger Type) e, phi_pq
Step2: Set e = 0
Step3: Set phi_pq = q-1
Step4: Set phi_pq = phi_pq* (p-1)

```
    Step5:      Set int i = 0
    Step6:      do {
                    Set e = (call BigInteger (bitsize, 0, new
Random ())).setBit (0)
    /*setBit () is used to set one bit to a bit string at the specified
position with the specified bit value. New Random () gives a
random number and it creates a new random number generator.
Its seed is initialized to a value based on the current time*/
                Set i = i + 1
                } Repeat Step6 while (i<100 && gcd (e,
phi_pq)! =1)
    /* we hope to find e with in 100 attempts. In each attempt
we try with different random BigInteger variable.*/
    Step7:      return e
```

Function gcd (int x)
It returns greatest common divisor among the instance variable e, that it has been called upon and the variable x.

The workings of this algorithm are well known and therefore needs no further description.

Function BigInteger calculate_d (BigInteger p, BigInteger q, BigInteger e)
This algorithm takes input as two large prime numbers p and q and the public exponent e of RSA and returns the private exponent d of RSA which is multiplicative inverse of e under the group of multiplication modulo n [23]where n = (p-1)*(q-1).

```
Step 1:  Declare BigInteger d, phi_pq
Step2:  Set phi_pq = q-1
Step3:  Set phi_pq = phi_pq * (p-1)
Step4:  Set d = e.modInverse (phi_pq)  //e.modInverse (m) returns
(e-1 mod m).

Step5:  return d
```

Function BigInteger rsa_encrypt (BigInteger msg, BigInteger e, BigInteger n):

```
Step1:  Declare BigInteger c, bitmask
Step2:  Set c = 0   //big integer type
Step3:  Set int i = 0
Step4:  Set bitmask = (2) ^ (n.bitLength ()-1)-1)
Step5:  while (msg.compareTo (bitmask) == 1) {
        Set c = msg.and (bitmask).modPow (e, n).shiftLeft (i*n.
bitLength ()) |c
        Set msg = msg.shiftRight (n.bitLength ()-1)
        Set i = i+1
        }

Step6:  Set c = msg.modPow (e, n).shiftLeft (i*n.bitLength ()).
or(c)
Step7:  return c
```

Function BigInteger rsa_decrypt (BigInteger crypt, BigInteger d, BigInteger n)

```
Step1: Declare Big Integer msg, bitmask
Step2: Set msg = new BigInteger ("0")
Step3: Set int i = 0
Step4: Set bitmask = ((2) ^ (bitLength of n))-1
Step5: while (crypt.compareTo (bitmask) == 1) {
/*The well-known function compareTo () returns less than zero
if the invoking object is lesser than the object that passed as the
argument according to the Dictionary order of the constituent
letters of the string objects.*/
Set msg = crypt.and (bitmask).modPow (d, n).shift Left (i*(n.
bitLength ()-1)).or (msg)
/*and () does the bit wise and operation; modPow (d, n), if invoked
on x, returns x^d mod n
x.or(y) gives the bit wise OR value of x and y.*/
Set crypt = crypt.shiftRight (n.bitLength ()) //shifting n bits to
the right hand side.
Set i = i+1
}

Step 6: Set msg = crypt.modPow (d, n). shiftLeft (i*(bit length
of n-1)).or(msg)
//x.or(y) gives the bit wise OR value of x and y.

Step 7: return msg
```

IV. Conclusion

Using this algorithm we can encrypt and decrypt user and/or application data very easily and the simplicity of this algorithm is the soul of this algorithm though it creates combinational complexity for a eavesdropper to decrypt the cipher-text but for end users this algorithm never poses any complex functional activity to perform. Without knowing the Diffie-Hellman Key decryption of the cipher in this system is analyzed further more. Finding one digit of the actual byte code means finding one specific addend and augends for a specific number ranging from zero to up to nine in the Addition modulo ten arithmetic (see Table1). Now, as it appears in Table 1 that any single digit from zero to up to nine may have ten different combination of different Addend and Augend.Therefore probability of finding one such specific digit is 0.1 and for n such digits this probability will be 1/(10)n . It should be noted that n is the size of the byte code of the message. For smaller sized data segment it is even more difficult because remember for smaller sized data we used RSA encryption first and on top of that we employed SecureByteConversion (Byte code) algorithm (see Step 2 of the SecureByteConversion algorithm). The most beneficiary part of it is that two different set of keys (one Diffie-Hellman key set and the other RSA key set) are used to encrypt the data/byte stream, which in other words makes life uncomfortable for a hacker to decrypt the cipher text. The hacker has to try all sort of combination of the RSA and Diffie-Hellman key to find the exact combination for the specific transmission. Moreover, as we did not disclose the Public Components of Diffie-Hellman Part (i.e. generator and the prime modulus) the eavesdropper must have to break the RSA first to only find them and then have to counter to find Diffie-Hellman Secret key.

References

[1] Adleman L. M.,"Factoring numbers using singular integers", The twenty-third annual ACM symposium on Theory of computing, New York, USA, p. 64, 1991.

[2] Boneh D.,"Twenty Years of Attacks on the RSA Cryptosystem", Notices of the AMS, Vol. 46, No. 2, pp. 203, 1999.

[3] Shor P. W.,"Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, pp. 124, 1994.

[4] Abadi M., Rogaway P.,"Reconciling two views of cryptography (the computational soundness of formal encryption)", J. of Cryptology, Vol. 15, No. 2, pp. 103, 2002.

[5] Lamacchia B. A., Odlyzko A. M., "Computation of Discrete Logarithms in Prime Fields", Designs, Codes and Cryptography, Vol. 1, pp. 47, (Springer), 1991.

[6] Bleichenbacher D., Kaliski B., Staddon J.,"Recent Results on PCKS: RSA Encryption Standard", RSA Laboratories' Bulletin, Number 7, 1998.

[7] Diffie W., Hellman M.,"New Directions in Cryptography", IEEE Transactions on Information Theory, Vol. 22, pp. 644, 1976.

[8] Shannon C. E.,"Communication theory of secrecy systems," Bell Syst. Tech. J., Vol. 28, pp. 656, 1949.

[9] Rivest R.L., Shamir A., Adleman L.,"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Magazine Communications of the ACM, Vol. 21, pp. 120, 1978.

[10] Denzer V., Ecker A.,"Optimal Multipliers for Linear Congruential Pseudo-Random Number Generators with Prime Moduli". BIT Numerical Mathematics, Vol. 28, pp. 803. (Springer) 1988.

[11] Stern J.,"Advances in Cryptology EUROCRYPT'99", vol. 1592 of Lecture Notes in Computer Science, pp. 223. (Springer-Verlag), 1999.

[12] Koblitz N., Menezes A. J.,"A Survey of Public-Key Cryptosystems", SIAM Review, Vol. 46, pp. 599, 2004.

[13] Nicolosi A., Krohn M., Dodis Y., Mazi`eres D.,"Proactive Two-Party Signatures for User Authentication", The 10th Annual Network and Distributed System Security Symposium, San Diego, California, 2003.

[14] Pointcheval D., Stern J.,"Security Arguments for Digital Signature and Blind Signature", Journal of Cryptology, Vol. 13, pp. 361, 2000.

[15] Rivest R. L., Kaliski B.,"RSA Problem", Encyclopedia of Cryptography and Security (Kluwer), 2003.

[16] Kurosawa K., Desmedt Y.,"A New Paradigm of Hybrid Encryption Scheme", Advances in Cryptology – CRYPTO 2004, Lecture Notes in Computer Science, Vol. 3152, 345. (Springer), 2004.

[17] Bellare M., Boldyreva A., Adriana Palacio,"An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem", Lecture Notes in Computer Science, Vol. 3027, pp. 171, (Springer) 2004.

[18] Shoup V.,"Session-key distribution using smart cards, Proc. Eurocrypt '96, pp. 321. "A note on session-key distribution using smart cards, manuscript". This contains some corrections and modifications to the previous paper, 1996. [Online] Available: http://www.shoup.net/papers/smartcards.pdf. [Online] Available: http://www.shoup.net/papers/update.pdf

[19] Canetti Ran, Krawczyk Hugo,"Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels", Advances in Cryptology — EUROCRYPT 2001 Lecture Notes in Computer Science, Vol. 2045, pp. 453. (Springer.), 2001.

[20] RogawayP., Shrimpton T.,"Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance and Collision Resistance", Appears in Fast Software Encryption (FSE 2004), Lecture Notes in Computer Science, (Springer-Verlag), 2004.

[21] Murdoch S.J., Drimer S., Anderson R., Bond M.,"Chip and PIN is Broken" IEEE Symposium on Security and Privacy (SP), 16-19 at Oakland, CA, USA ,pp 433, May 2010.

[22] Pomerance C.,"The Number Field Sieve", The development of the number field sieve Lecture Notes in Mathematics, Vol. 1554, (Springer), 1993.

[23] Alsultanny Y. A.,"Image Encryption by Cypher Feedback Mode", International Journal of Innovative Computing, Information and Control, Vol. 3, pp. 589, 2007.

[24] Bellare M., Kilian J., Rogaway P.,"The security of cipher block chaining" Advances in Cryptology — CRYPTO '94 Lecture Notes in Computer Science, Volume 839, pp 341 (Springer), 1994.

Sahnoaj Ahmed was born in 1982. He received his BTech in Computer Science & Engineering from University of WBUT in 2005 and MTech from WBUT in 2010.He has 10 years of experience in teaching. His research interest is in Cryptography and Image Processing.

Bijoy Kumar Mandal, he is, currently, associated with Computer Science and Engineering Department, Faculty of Engineering and Technology, NSHM Knowledge Campus – Durgapur, as an Assistant Professor. He is pursuing Ph.D. (Computer Science and Engineering) in NIT, Durgapur. He published 26 Research papers in international Journals and Conferences and two books National & International. He had more than 8 years teaching & research experience.

Arindam Biswas was born in West Bengal, India in 1984. He received M-Tech degree in Radio Physics and Electronics from University of Calcutta, India in 2010 and PhD from NIT Durgapur in 2013. He was a Post-Doctoral Researcher at Pusan National University, South Korea with prestigious BK21PLUS Fellowship, Republic of Korea. Arindam Biswas has 8 years' experience in teaching research and administration. He has more than 50 papers and 4 books with international repute. His research interest is in carrier transport in low dimensional system and electronic devices and non-linier optical communication and bioinformatics. One PhD is awarded under his joint guidance with NIT, Durgapur

Anup Kumar Bhattacharjee received his BE in Electronics and Telecommunication Engineering from BE College Shibpur, Howrah in 1983. He received his ME and Ph.D. from Jadavpur University, Kolkata in 1985 and 1989 respectively. Presently he is attached with Electronics and Communication Engineering Department, in National Institute of Technology, Durgapur, and West Bengal, India as a Professor. His area of research is in Microstrip Antenna, Embedded System, and Mobile Communications etc. fourteen number of PhD successfully awarded under his guidance.