ISSN: 2229-6948(ONLINE)
DOI: 10.21917/ijct.2012.0093

DMAC-AN INTEGRATED ENCRYPTION SCHEME WITH RSA FOR AC TO OBSTRUCT INFERENCE ATTACKS

R. Jeeva
Department of Computer Science and Engineering, Joe Suresh Engineering College, India
E-mail: r.jeevanatham@gmail.com

Abstract
The proposal of indistinguishable encryption in Randomized Arithmetic Coding(RAC) doesn't make the system efficient because it was not encrypting the messages it sends. It recomputes the cipher form of every messages it sends that increases not only the computational cost but also increases the response time.Floating point representation in cipher increases the difficulty in decryption side because of loss in precison.RAC doesn't handle the inference attacks like Man-in-Middle attack,Third party attack etc. In our system, Dynamic Matrix Arithmetic Coding(DMAC) using dynamic session matrix to encrypt the messages. The size of the matrix is deduced from the session key that contains ID of end users which proves the server authentication.Nonce values is represented as the public key of the opponents encrypted by the session key will be exchanged between the end users to provide mutual authentication. If the adversary try to compromise either server or end users,the other system won't respond and the intrusion will be easily detected. we have increased the hacking complexity of AC by integrating with RSA upto 99%.

Keywords:
RAC, DMAC, Hacking Complexity, Third Party Attack, Inference Attack

1. INTRODUCTION

Arithmetic coding - The source symbols are encoded numerically [1] [6]. Each symbol does not necessarily translate into the same indexed code each time it is encoded. An input source string is represented by an interval of real numbers between 0 and 1. The range of the interval is need by two values, high and low, which are equal to 1 and 0 initially. The interval is successively subdivided as each new source symbol is read. Highly probable symbols (with respect to the model, as distinct from the more probable symbols in the input) reduce the interval by a smaller amount than less probable symbols. The precision with which to represent the interval increases in accordance with the length of the input string.

Key Based Interval Splitting-a modification of arithmetic coding that uses key-based interval splitting to simultaneously enable data compression and encryption [2][7]. We have shown that even when intervals in an arithmetic coder are split, the code length increases relative to traditional arithmetic coding is bounded to less than 1 bit per N-symbol sequence, and in practice,the increase is often approximately 0.5 bits per -symbol sequence. In percentage terms, this efficiency penalty becomes negligibly small as N increases. The splitting produces encryption, the level of which is a function of the attributes of the key and the encoded sequence. While we have focused on the static binary case for simplicity, the methods presented here can also be applied to M-ary and/or adaptive arithmetic coding.

Secure Arithmetic Coding-An arithmetic coder in which the intervals associated with each symbol combination are split in accordance with a key, and in which permutations are applied both to input symbol sequence and to the output binary sequence, has been presented [3][9]. The system offers both compression and security, and thwarts all known attacks aimed at obtaining information about the input or output permutation or the interval splitting keys. For each encoded symbol, a pair of intervals is split, and this split can occur in parallel. So the throughput can be identical to that of a traditional arithmetic coder. The permutations add negligible complexity. The security problem of Secure Arithmetic Coding (SAC) under an adaptive chosen-cipher text attack is it can recover the key vectors used in the codeword permutation step with complexity O (N), where N is the symbol sequence length [4]. This indicates that the SAC is not suitable for those applications where the attacker can have access to the decoder. Furthermore, we have discussed an improved version of the SAC such that it can resist the adaptive chosen-cipher text attack and can be conveniently incorporated with the context-based coding.

Modifications of arithmetic coding (AC) have been proposed to improve the security of traditional AC [8] [10]. Two main modifications to AC are randomized AC (RAC) and AC with key-based interval splitting (KSAC). Chosen-plaintext attacks have been proposed for these two methods when the same key is used to encrypt different messages [9]. The security of AC in is based on the inability of the adversary to distinguish between the encryption of one plaintext from the encryption of another. By this definition, we prove that RAC is insecure even if a new random key is used to compress every message. The adversary can only eavesdrop on the cipher texts and cannot request encryptions of chosen-plaintexts. The method of first-compress-then-encrypt, where the encryption is performed by a bitwise xor of the compressed output with a pseudorandom bit sequence, is provably secure with respect to chosen-plaintext attacks[13].

In our system, we increse the security of arithmetic coding by merging the DMAC, dynamic sized session matrix with RSA algorithm. We handle the communication line attacks like Man-in-Middle attack, Third party attack etc by introducing nonce values between the end users that provides mutual authentication and check the server authentication using the public key of end users.

2. EXISTING SYSTEM

In this scheme (RAC) [4] that is chosen-plaintext secure, this is a stronger notion of security than having indistinguishable encryptions in the presence of an eavesdropper. To understand this scheme, it looks at AC as defined in, where the interval [0, 1) is split in two ways as shown in Fig.1. The traditional way of partitioning the interval [0, 1) is according to the probabilities (p_A, p_B) and the subintervals are labeled with symbols A and B.

Another way to partition the interval [0, 1) is split into subintervals of equal lengths and labeled 0 and 1. Partitioning once results in two subintervals [0, 0.5) (labeled 0) and [0.5, 1) (labeled 1). Partitioning once more results in each subinterval being partitioned into two intervals of equal length. Thus the interval [0, 1) is partitioned into the following four subintervals of equal length: [0, 0.25) (labeled 00), [0.25, 0.5) (labeled 01), [0.5, 0.75) (labeled 10), and [0.75, 1) (labeled 11). Therefore, doing the partitioning n times results in the interval [0, 1) being partitioned into 2^n subintervals each labeled with a distinct -bit binary number. The i^{th} of these subintervals is referred to as $E_i(n)$ and label $E_i(n)$ as the n-bit binary expansion of i, that is the label for that subinterval ($i = 0,1,\ldots.2^{n-1}$). A string S that needs to be compressed is first converted into interval I(S) using the traditional AC method. Then the smallest n such that there exists $E_i(n)$ and that is completely within I(S) is found (or $E_i(n)$ subset of I(S)). The n-bit label on this $E_i(n)$ is the output of the compression algorithm. The bits of the output can be put out as the symbols S of are being read[11].

Let S = ABBAB and pA = 2/3 = 0.67 and pB = 1/3 = 0.33. Using traditional AC we find that after A is read, I (A) = [0.33, 1). If there exists an n and I such that I (A) is completely within $E_i(n)$. $E_0(n)$ = [0, 0.5) and $E_1(n)$ = [0.5, 1). Since I(AB) is not within $E_0(n)$ or $E_1(n)$, no output can be put out yet[17]. The next symbol B is read and I (AB) = [0.33,0.55). Again, since I(AB) is not within $E_0(n)$ or $E_1(n)$, no output can be put out yet. The third symbol B is read and I(ABB) = [0.33,0.4026). Now I(ABB) is within $E_1(1)$ = [0.25,0.5) with label 01. Therefore the output so far is 01. This is true because any interval $E_i(n)$ that is within I (ABB) is a subinterval of $E_1(2)$. Note that I (ABBA) is not within any of these intervals so the next symbol B is read. I (ABBAB) = [0.354, 0.37). I(ABBAB) is within $E_2(3)$ = [0.25,.375) and the output is the 3-bit binary value of 2 which is 010[11]. Ei(3), i = 0,…..7, consists of the intervals [0,1/23),[1/23,2/23),,,,,[1/23,1)=[0,0.125),[0.125,0.25),[0.25,0.375),[0.375,0.5),[0.5,0.625),[0.625,0.75),[0.75,0.875),[0.75,1).Since the bits 01have already had been output, we now output the third bit 0. However, I (ABBAB) is within a smaller interval, E5(4) = [5/24, 6/24) = [0.3125, .375). Since the 4-bit binary expansion of 5=0101 and the most significant three bits 010 have already been output we now output the next bit 1. Since the last symbol has already been read, we now look for i and smallest n such that $E_i(n)$ is completely within I(ABBAB).

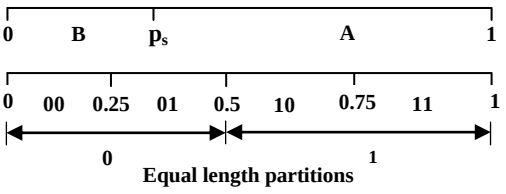

Fig.1. Two ways to partition the range [0, 1)

It finds that and i = 23 and n = 6. Thus, $E_i(n) = E_{23}(6) = [23/2^6, 24/2^6) = [0.3596, 0.375)$ is completely within I(ABBAB). Note that this is the largest interval $E_i(n)$ that is completely within I(ABBAB). Since the 6-bit binary expansion of 23 = 010111 and 0101 has already been output, we output the remaining bits 11. Thus the final output is 010111 and the decimal value of 0.010111 is 0.359375 and this is within I(ABBAB) as expected. Note that maintaining $E_i(n)$ can be done incrementally with minimal overhead[11].

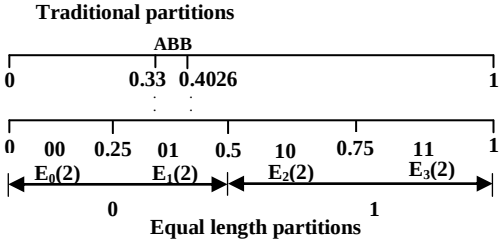

Fig.2. I (ABB) is a subinterval of $E_1(2)$

The above procedure as in Fig.2 was converted into an encryption by simply permuting the labels for each $E_i(n)$. This can be done by doing a bitwise XOR of each label with a pseudorandom bit sequence of length n. Consider the labels of $E_i(2)$, which are 00, 01, 10, and 11. Doing a bitwise XOR with a 2-bit pseudorandom bit sequence, say 10, with each of the $E_i(2)$ results in $E_1(2)$=10, $E_1(2)$=11, $E_2(2)$=00, $E_3(2)$=01[12]. This system is simply AC followed by XORing with a pseudorandom bit sequence using randomized counter mode of operation. It looks at AC in a different way. The interval [0, 1) is split in two ways: 1) Split according to the probabilities p_A and p_B , and 2) split into intervals of equal length. In RAC, randomization is done using the first way of splitting [0, 1), while in our method randomization is done using the second way of splitting [0, 1). Since the randomization or permutation of the intervals is performed on intervals of equal length, unlike that in RAC, the new scheme is secure.

3. PROPOSED SYSTEM

3.1 SYSTEM ARCHITECTURE

In our system in Fig.3, the security is provided in two ways: one by the server; other by the end user itself. When the user login the system, not only the status of the system is sent to the server, but also the public key of the user. During login, the end user generates two random prime numbers that in turn generates private and public key of the user. When the user chooses the recipient, the server checks the status of the recipient. If it is in Active state, it generates a session key that contains the details of sender ID, receiver ID, sender public key and recipient public key. When the sender and receiver receive this session key, they can verify the received public key with their own public key that provides authentication between the server and end users. The sender uses the receiver's public key to encrypt the entries in the session matrix. Then the nonce values, public key of the end users encrypted with the received session key provided by the server will be exchanged between the end users. The public key of the receiver is encrypted with the session key sent to the recipient by the sender. The receiver verifies its own public key and in turn sends the sender's public key and the sum of sender and receiver public key encrypted with session key send to the sender. This process provides the mutual authentication between the end users.

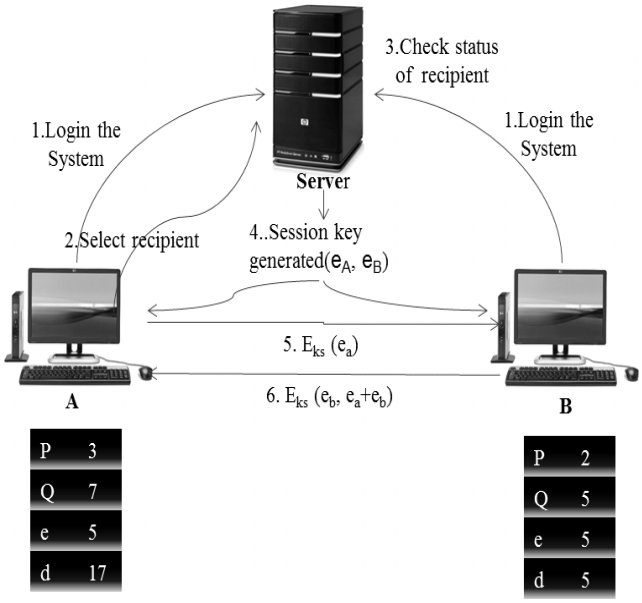

Fig.3. Architecture of DMAC

3.2 MATRIX SIZE DETERMINATION

After getting the session key from the server, it undergoes the sum of the summation of all the ASCII Codes in the message and its corresponding base value as in Fig.4. The output of the step-1 is used in two eays: one to generate a row value and the other to generate the column value. In row compuation side, the sum of the product of each digit of the ouput value in step-1 and the place value of each digit. The ouput value from step-1 taken modulo with the ouput value from step-3 to give the Row Size.In column compuation side, the sum of the product of each digit of the ouput value in step-1 and the reverse place value of each digit. The ouput value from step-1 is taken modulo with the ouput value from step-5 to give the Column Size.

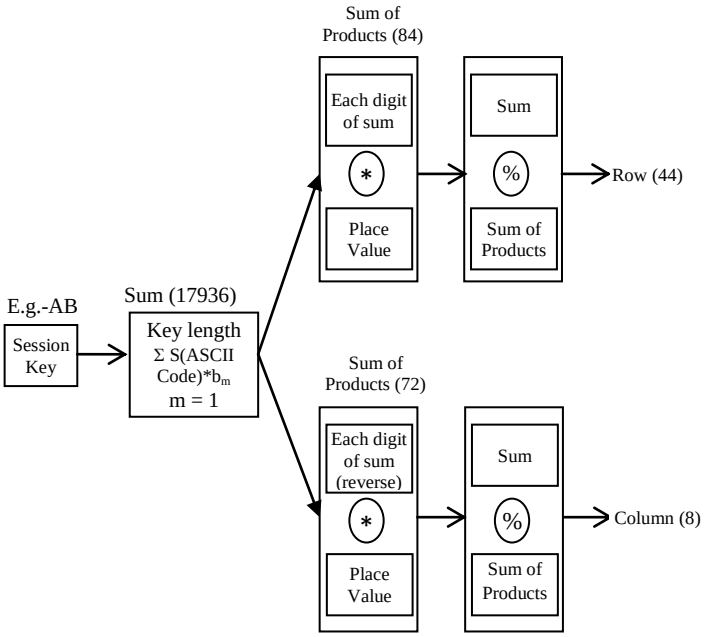

Fig.4. Size Deduction

3.3 MATRIX ENCRYPTION USING RSA

After computing the row and column size, we formulate the dynamic session matrix with the above computed value. The entry of the matrix is filled with the sequence of numbers starting from 1, continues till it reach the entry for a[n][n] as shown in Fig.5. As per the above architecture, the sender holds the public key of the receiver and vice versa. The user who creates the matrix to send the message encrypt the entries of the matrix as M with the public key of the opponent using RSA Algorithm. Receiver will decrypt the values using his private key.

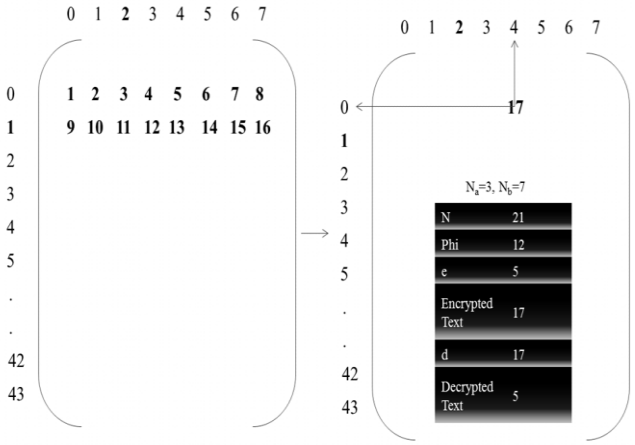

Fig.5. Entry Encryption with RSA

3.4 ENCRYPTION-DMAC

After the message formation,the message is converted into ASCII form. Each character's ASCII form is added with session key length provided by the server. The above output is split into nth and n - 1 digits that taken as row index and column Index.Using the above computed indexes, get the entry from the encrypted matrix computed in previous process. Repeat step-3 and step-4 for every charactes in the message. Calculate the hash value of the original message and append with the above value to form a resultant cipher as shown in Fig.6.

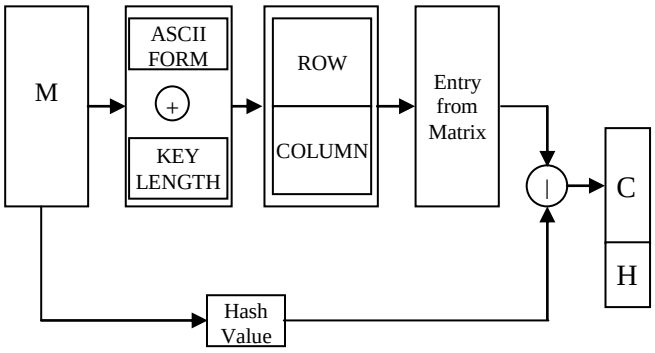

Fig.6. Encryption using DMAC

3.5 DECRYPTION-DMAC

In Fig.7, the cipher and the hash value is separated after it is received.The cipher value is compared with the encrypted matrix using sender's public key. The Matching indexes will be retrieved and stored in the location A. The same cipher value is

decrypted with the receiver's private key,the resulting value is compared with the base matrix. The Matching indexes will be retrieved and stored in the location B. If A = B,then compute the subtraction of key value from A or B which provides confidentiality. Perform the reverse ASCII process and compute the hash value. If the received hash value equal to computed hash value, then the message is received as send by the sender.

3.6 OBSTRUCTION OF MAN IN THE MIDDLE & THIRD PARTY ATTACK

In this system, it shows the sign in process of client. The server is in ready state to receive the client requests. As usual client passes the user name and password encrypt with server current public key known only to the legitimate server and client. Access will be granted by returning the current client public key which is known only to the corresponding client and trusted server if the above condition is true. If Man-in-Middle attack or third party attack rises, it has the possibility of giving fake privileges but not the current public key of the user. At the same time if the client that act as adversary tries to reroute the messages to original server, the server will detect the difference between the ID in the request and public key and adversary can be blocked.

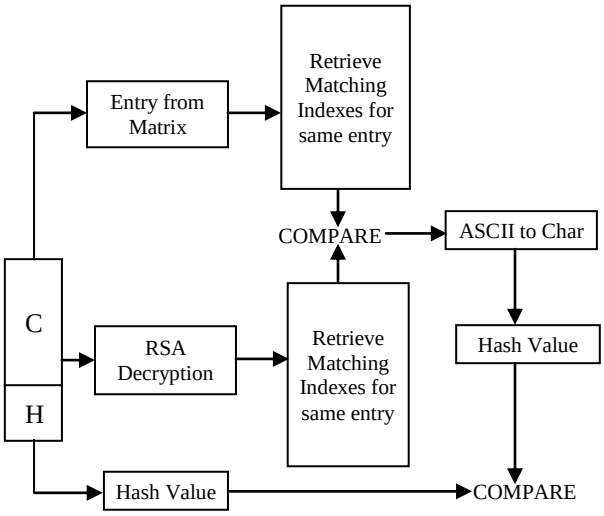

Fig.7. Decryption using DMAC

3.7 ALGORITHM-DMAC

```
Function Matrix_Size()
begin
        n1= session key. length;
        n2= base (n1);
        for i = 0 to n1 do
          n3=ASCII (key[i])*n2^i;
                for i = 1 to n3.length do
                n4 = n3[i]*i;
                    for i = n3.length to 1 do
                    begin
                        j = 0;
                        n4 = n3[j]*i;
                        j++;
                    end
          rowIndex = n3%n4;
          colIndex = n3%n5;
end

Function DMACEncrypt()
begin
        m1= ASCII(m)+sessionkey.length;
        m2= nth term of m2;
        m3= (n-1) terms of m2;
        c1= c[m2][m3];
        h1= hash(m);
        append c1 with h1 as Cipher
        return Cipher
end
```

3.7.1 Description:

Get the message to be send. Convert into ASCII form and adding session key length provided by server. Encrypt the matrix using recipient's public key to encrypt the message. Get the entries for the corresponding indices from the matrix. On decryption, decrypt the cipher with receiver private key and find the indexes match with the base matrix. Match the encrypted cipher with the encrypted matrix to get the indexes and compare both the indexes. Compute the hash value and check with the received hash value to prove the data integrity.

4. PERFORMANCE ANALYSIS

The key vectors used in the code word permutation step (Existing SAC) can be recovered with complexity $O(N_C)$ where is N_C code word length. The Table.1 shows the comparative study of hacking complexity between the AC, ESAC [5], RMAC and DMAC. In Fig.8, based on the comparative study DMAC has the higher hacking complexity compared to other algorithms.

Table.1. Hacking Complexity between ESAC & DMAC

Text Length	Code Word				Hacking Complexity				Performance		
	AC	ESAC	RMAC	DMAC	AC	ESAC	RMAC	DMAC	ESAC	RMAC	DMAC
100	100	137	119	102	100	416	241	105	24.038	41.493	95.238
200	200	237	219	202	200	716	441	205	27.932	45.351	97.560
300	300	337	319	302	300	1016	641	305	29.527	46.801	98.360
400	400	437	419	402	400	1316	841	405	30.395	47.562	98.765
500	500	537	519	502	500	1616	1041	505	30.940	48.030	99.009
600	600	637	619	602	600	1916	1241	605	31.315	48.348	99.173
700	700	737	719	702	700	2216	1441	705	31.588	48.577	99.290
800	800	837	819	802	800	2516	1641	805	31.796	48.750	99.378
900	900	937	919	902	900	2816	1841	905	31.960	48.886	99.447

4.1 SAMPLE CALCULATION

Text Length (T) = 700
Destination user name = 8-16
Key Range = 8-16
Maximum key Range = 16
Maximum Destination user name (D) = 16
Random No (NR) = 3
Port Identity (PI) = 1
Hash value (H) = 1
No. of Control Messages by client (CM) = 2
No. of Control Messages by server (SM) = 1

Total length of Generated Cipher (RMAC)
= T + K + D + NR + PI + H
= 700 + 16 + 16 + 3 + 1 + 1
= 737

Hacking Complexity (RMAC) = O(Nc)
= 700

Total length of Generated Cipher (DMAC)
= T + NR + H
= 700 + 1 + 1
= 702

Hacking Complexity (DMAC) = O(Nc) + 3
= (702) + 3
= 705

$$\text{Performance} = \frac{\text{Hacking Complexity (RMAC)}}{\text{Hacking Complexity (DMAC)}} * 100$$

$$= \frac{700}{705} * 100$$

$$= 99.29\%$$

The security of the AC is increased by our system by 99%.

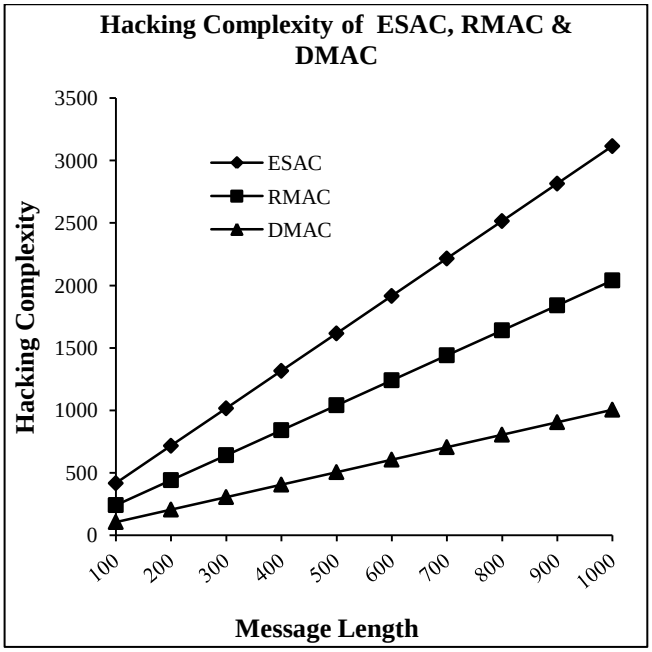

Fig.8. Hacking Complexity between ESAC, RMAC and DMAC

5. CONCLUSION

In our system, we satisfied 3 out of 4 goals of cryptography that are data integrity, confidentiality and authentication. Usage of integer form with randomness in representing cipher, not only makes the decryption easy but also it reduces the overhead in computation in handling precision. Message indistinguishability is not needed because the authentication is provided both from the server and end users. The unsatisfied goal is nothing but non-repudiation that requires logs which records the activities during the session that leads to log management, concurrency control, database security etc that makes the system very complex to implement.

REFERENCES

[1] H. Kim, J.T. Wen and J.D. Villasenor, "Secure arithmetic Coding", *IEEE Transaction on Signal Processing*, Vol. 55, No. 5, pp. 2263-2272, 2007.

[2] J.T. Wen, H. Kim and J.D. Villasenor, "Binary arithmetic coding with key-based interval splitting", *IEEE Signal Processing Letters*, Vol. 13, No. 2, pp. 69-72, 2006.

[3] Jiantao Zhou, Oscar C. Au and Peter Hon-Wah Wong, "Adaptive chosen-cipher text attack on secure arithmetic coding", *IEEE Transaction on Signal Processing*, Vol. 57, No. 5, pp. 1825-1838, 2009.

[4] Raj S. Katti, Sudharsan K. Srinivasan and Aida Vosughi, "On the Security of Randomized Arithmetic Codes Against Ciphertext-Only Attacks", *IEEE Transaction on Information Forensics and Security*, Vol. 6, No. 1, pp. 19-27, 2011.

[5] V. Kavitha and S. Balaji, "ESAC Based Channel Aware Routing with Route Handoff", *International Journal on Computer Science and Engineering*, Vol. 3, No. 3, pp.1260-1269, 2011.

[6] R. Bose and S. Pathak, "A novel compression and encryption scheme using variable model arithmetic coding and couple chaotic system", *IEEE Transactions on Circuits and Systems*, Vol. 53, No. 4, pp. 848-857, 2006.

[7] P.W. Moo and X. Wu, "Resynchronization properties of arithmetic coding", *Proceedings of the IEEE International Conference on Image Processing*, Vol. 2, pp.545-549, 1999.

[8] H.A. Bergen and J.M. Hogan, "Data security in a fixed-model arithmetic coding compression algorithm", *Computers and Security*, Vol. 11, No. 5, pp. 445-461, 1992.

[9] Lan H. Witten and J.G. Clearly, "On the privacy offered by adaptive text compression", *Computers and Security*, Vol. 7, No. 4, pp. 397-408, 1988.

[10] H.A. Bergen and J.M. Hogan, "A chosen plaintext attack on an adaptive arithmetic coding compression algorithm", *Computers and Security*, Vol. 12, No. 2, pp. 157-167, 1993.

[11] H. Ishibashi and K. Tanaka, "Data encryption scheme with extended arithmetic coding", *Proceedings of the SPIE, Mathematics of Data/Image Coding, Compression and Encryption IV with Applications* , Vol. 4475, pp. 222-233, 2001.

[12] A.J. Menezes, P.C. Van Oorschot and S.A. Vanstone, "*Handbook of Applied Cryptography*", CRC Press, 1997.

[13] P.G. Howard and J.S. Vitter, "Arithmetic coding for data compression", *Proceedings of the IEEE*, Vol. 82, No. 6, pp. 857-865, 1994.