ESCORTING DATA FROM CCA2 ATTACK ON SAC

Balaji S., Jeeva R., Ayyam Perumal M., Mano P.

SCAD College of Engineering & Technology

ABSTRACT

The improved system of SAC prevents many attacks like resisting adaptive chosen-cipher text attack and can be conveniently incorporated with the context-based coding. So the attacker still cannot reduce the uncertainty of which symbol is splitted.But it doesn't talk about detecting any alteration or loss in data in the middle of the transmission and the person's authenticity that is sending and receiving the data involved in the transmission is missing. In our system, improved SAC are escorting SAC by adding information about sender and preserves the message content by computing hash function which prevents not only attacks mentioned previously but also maintains the integrity of data.

I. INTRODUCTION

Arithmetic coding is a popular and efficient lossless compression technique that maps a sequence of source symbols to an interval of numbers between 0 and 1 [1]. The encoder produces a code stream of bits that uniquely represents the interval; the decoder then maps the code stream to the original source sequence. In arithmetic coding, an entire source sequence is mapped to a single code stream. Therefore, a single error in an arithmetic code stream often causes error avalanches at the decoder, rendering the decoded code stream useless. *Full resynchronization* occurs when the decoder can exactly determine the initial *b* bits of the code stream. In this case the entire original source sequence can be reproduced exactly. *Partial resynchronization* occurs when the decoder only determines the current interval after b bits of the code stream.

Binary arithmetic coding involves recursive partitioning the range (0, 1) in accordance with the relative probabilities of occurrence of the two input symbols [2]. The overall length within the range (0, 1) allocated to each symbol is preserved, but the traditional assumption that a single contiguous interval is used for each symbol is removed. A key known to both the encoder and decoder is used to describe where the intervals are "split" prior to encoding each new symbol. a key-based interval splitting arithmetic coder can be implemented using techniques similar to those used in traditional arithmetic coding and can benefit from the same optimizations for speed, finite precision, etc. The main difference lies in the doubling of the number of intervals, which doubles the memory requirement because the upper and lower limits of two intervals must be maintained. The number of potential split locations—and therefore the level of secrecy— are determined in part by the precision of the key. The key can be used to directly identify split locations, or it can reference locations in a table known to both the encoder and decoder. The splitting produces encryption, the level of which is a function of the specific attributes of the key and the encoded sequence.

A chaos-based adaptive arithmetic coding-encryption technique has been designed, developed and tested and its implementation has been discussed[3]. For typical text files, the proposed encoder gives compression between 67.5% and 70.5%, the zeroth-order compression suffering by about 6% due to encryption, and is not susceptible to previously carried out attacks on arithmetic coding algorithms. Characteristics of chaotic systems like ergodicity, mixing and sensitivity to initial conditions have been seen as analogous to and/or giving rise to confusion and diffusion balance and avalanche property , known properties of a good cipher. The proposed algorithm is resistant to chosen plaintext attacks because of the following. The model dynamically reorders the frequency of the input symbols according to the coupled chaotic system, and depends on all text that has been coded since the initialization of the model. The model dynamically reorders the frequency of the input symbols according to the coupled chaotic system, and depends on all text that has been coded since the initialization of the model. The output from the engine is in the form of variable sized words and the individual bit output corresponding to inserted symbols cannot be determined.

For SAC, a series of permutations are applied at the input and the output of the encoder[4]. The overall system provides simultaneous encryption and compression, with negligible coding efficiency penalty relative to a traditional arithmetic coder. The system consists of a first permutation step applied to the input sequence, arithmetic coding using interval splitting, and a second permutation step applied to the bits produced by the coder. A key sequence is input to a key scheduler which in turn provides key information to both permutation steps and to the interval splitting arithmetic coder. The key scheduler utilizes information from the split AC encoder output. 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.

Compared with the original SAC,[5] improved SAC remove the input symbol permutation step. In addition, improved SAC replace the output codeword permutation step with a simple bit-wise XOR step. The design of the key scheduler is very flexible; we can either use the keyed XOR operation as in or other highly efficient pseudorandom number generators. The only private information of the improved system is the seed used in the key scheduler, which is assumed to be of length 128 bits, so as to ensure high enough level of security. It resists the adaptive chosen-cipher text attack and can be conveniently incorporated with the context- basedcoding.

In the present paper, we have identified the major flaw in the previous algorithm that is it deals with the security of data before transmission. That is when the attacker saw the codeword he shouldn't get nothing from the codeword. Any cryptography concept must satisfy its own four goals that are confidentiality, authentication, Data integrity and non-repudiation. So we going to make the SAC to satisfy these goals to escort the data from known attacks and the attacks happened in the transmission like Man-in-Middle attack, Replay attack etc.But while in the transmission it didn't talk or assured about detecting the data alteration or data loss and points the intruder who involved in the transmission unauthorized. We have framed our architecture by adding our system before and after the SAC process which transforms the text into numbers and the preservation and data integrity is maintained by adding hash function and sender's identity during and after the transmission.

II. REVIEW OF SAC

In the SAC two permutation steps are applied to the input symbol sequence and the output codeword, respectively. Let $S = s1, s2, \ldots sN$ be the symbol sequence to be encoded. The encoding procedure of the SAC is shown as follows [5].

Step 1 Map the sequence S into a block having four columns and $N/4$ rows.

Step 2 Perform two key-driven cyclical shift steps to the resulting symbol block, and read out the data in raster order to obtain the permuted symbol sequence $S1$.

In Fig. below, we show an example of the cyclical shift steps, where S is of length 16 and the key vectors controlling the column $CS = [2\ 3\ 0\ 1]$ and the row shift offsets $RS = [1\ 3\ 1\ 2]$ are and, respectively.

Step 3 Input to the ISAC encoder and obtain the intermediate codeword $C = c1, c2, \ldots\ldots\ldots cNc$.

Step 4 Set $C = c1, c2, \ldots\ldots\ldots cNc - 4$ by removing the last four bits of. Map into a block having four columns and $[(Nc - 4)/4]$ rows.

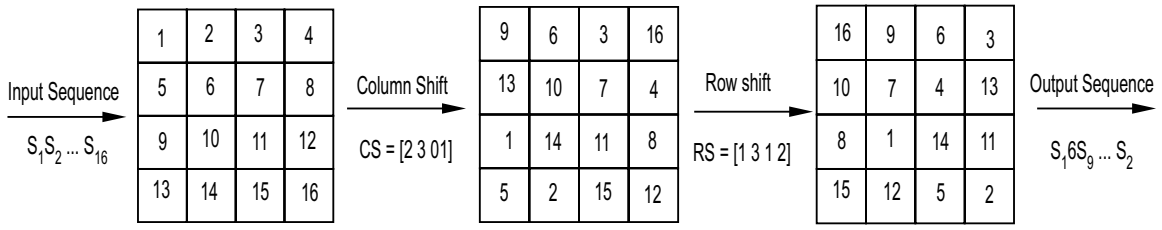

Fig. 1. Cyclical shift steps

Step 5 Perform the first round of shifts to the resulting bit block, which consists of two key-driven cyclical shift steps, one operating on columns and the other on rows. Here, the key vectors controlling the shift offsets depend on the last four bits of *C*.

Step 6 Reappend to the resulting bit block.

Step 7 Perform the second round of shifts to the resulting bit block, which consists of two key-driven cyclicalshift steps, one operating on columns and the other on rows. Here, the key vectors controlling the shift offsets are fixed for all.

Step 8 Read out the data in raster-order from the resulting block to obtain the final bit stream.

Improved SAC

The improved system which mainly consists of two parts:

1. an ISAC encoder
2. a key scheduler

Let the symbol sequence $S = s1, s2, \ldots sN$ be that is to be encoded. The basic steps of performing the encoding are as follows.

Step 1 Encode *S* using an ISAC encoder with splitting key *K* vector. Denote the generated bit stream $C = c1, c2, \ldots\ldots\ldots cNc$ as, where Nc satisfies (1).

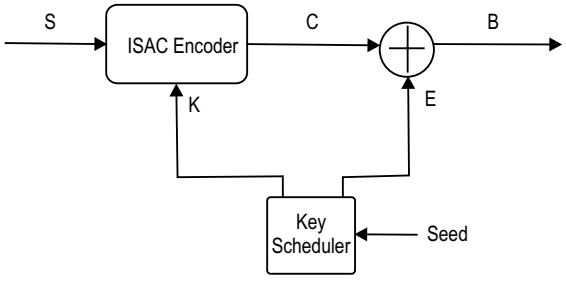

Fig. 2. Improved SAC

Step 2 Perform bit-wise XOR operation between *C* and a key stream *E*, where has the same length as *C*. In other words, the final bit stream $B = C \oplus E$

Compared with the original SAC, we remove the input symbol permutation step. In addition, we replace the output codeword permutation step with a simple bit-wise XOR step. The only private information of the improved system is the seed used in the key scheduler, which is assumed to be of length 128 bits, so as to ensure high enough level of security.

System Framework

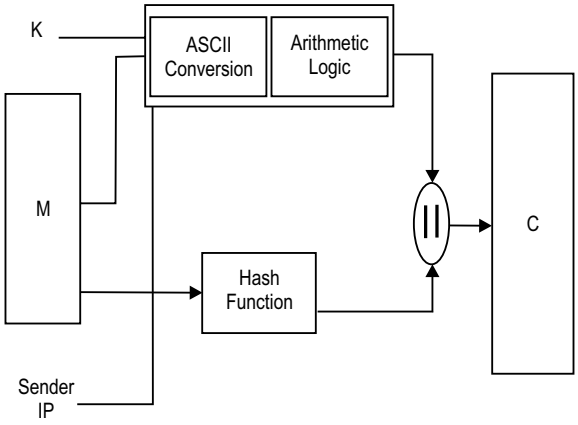

Fig. 3. Encryption

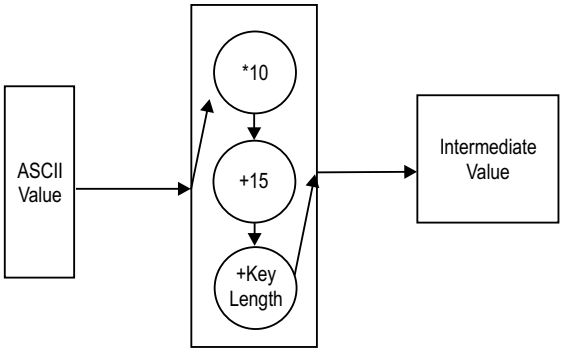

Fig. 4. Arithmetic Logic

Encryption

1. Create the message to be sent and give the key to encrypt the message.
2. Retrieve the sender's IP and calculates the hash code for the message.
3. Perform ASCII conversions for these three values and Fed ino Arithmetic logic.
4. In the Arithmetic logic, the ASCII code undergoes some arithmetic calculations and at the end key length is added.
5. Append a random value at the end of each parameter. For message the value ranges from 0 to 5. For key the value ranges from 6 to 10. For Sender ip the value ranges from 11 to 15.

6. After creating this codeword, append the hash value at the end of the cipher and transmits the message.

Centralization

1. The transmitted messages are received by the proxy and retransmitted to the intended client after checking its IP with its list.
2. It maintains a database which has the legitimate users IP along with their username and password.
3. Every time it receives the message, it retrieves the sender IP from the cipher and try to find the match with the list of IP stored in the database.

Decryption

1. After receiving the cipher from Proxy, retrieve the hash value from the cipher and store it in separate area.
2. Fed the remaining cipher into inverse arithmetic logic and perform text to ASCII conversion.
3. Separate the remaining three parameters and store in different area for easy retrieval.
4. Get the receiver key to decrypt and try for a match with the key retrieve from the cipher.
5. If it matches, then retrieve the sender IP and Check with the proxy. If it matches go for next checking task.
6. Then calculate the hash code for the decrypted message and try for a match with the hash code retrieve from the cipher.
7. If all the above conditions are satisfied, then displays the message to the receiver.

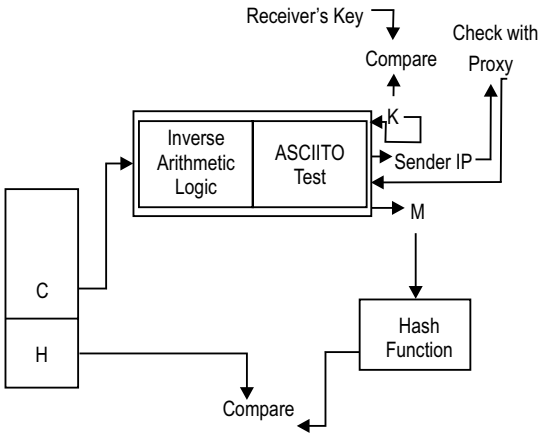

Fig. 5. Decryption

Message Creation

Make sure that server is running by getting the response from it to the client who wishes to start the transmission. After getting the response create the message to be transmitted by using ASCII key conversion and Arithmetic Logic. Then get the key known to both the sender and the receiver to encrypt the created message. Calculate the Hash code of the Original message and append to the above created cipher. Then encrypt the key as in the method message encrypted. Append the encrypted key to the above created cipher. Get the source ID of the client who is transmitting and destination of the client who is going to receive. The parts in the cipher like message, key, sender ID, destination ID and hash value of the original message are separated by random numbers in different ranges. After finishing all this process, transmit the message.

Monitoring Transmission

Centralised proxy is in ready state to receive the messages and to transmit the message. Once it receives the message, it strips the source ID and destination ID.Then it checks the source ID with the database stored in proxy for getting the match. If it matches precede the transmission and forward the message to the intended client. If it doesn't match with the ID stored in the database, deny the transmission of the client who just transmitted. And ask the client if he has an interest in registering this group. If he accepted, then ask the user to enter his details like username, password etc. After entering his username check its availability with the database stored in the centralised proxy. And then stipulates the user to choose the password between 8 to 16 characters. Once the details verified, Send the Acknowledgement for the accepted registration to the intended client.

Key Creation and Transmission

If the client wishes to transmit, he has to send the ready message to the centralised proxy. Proxy in turn generates a random key based on the client password who wishes to start the transmission. Send this random key to the client who is ready to transmitting. Then the sender in turn sends the key to the intended client. After receiving this message to the proxy, the sender key is encrypted with the intended client's password and forward the encrypted the key to the destination. Once the receiver receive this encrypted the key, using his password he can decrypt

the key transmitted by the sender. Then the communication starts. On further onwards all the messages are encrypted and decrypted with the temporary key known by both sender and receiver till the session has finished. Once the session is terminated the temporary key will be expired. For the next session another key will be generated for the same client.

Message Reception

Once the client receives the message from the proxy it decrypts the message with the receiver's key. After it decrypts it separates the parts of the cipher like original message, key, senders ID and the hash value of the original message. Compare the decrypted key with the entered receiver key. If it matches checks the sender ID with the database stored in the centralised proxy. If the result is Positive then calculate the hash value of the decrypted message and compares with the decrypted hash value of original message. If it matches displays the message to the receiver.

III. ALGORITHM

Input: M-Message, K-Key

Function encrypt () return value

Append random no (<5)

B[]=Alogic(IP,k.length)

Append random no (<10 & >5)

C[]=Alogic(k,k.length)

Append random no (<15 & >10)

Append hash (m)

Merge the arrays

Return merged array

Alogic(S,sk.length)

Begin

For i=0 to s.length do

A[]=ASCII(s)

A[]=10a[]+15+sk.length

end

IV. DESCRIPTION

1. Get Data to be sent.
2. Perform ASCII Conversion and Alogic to the above data.
3. Append Random no to the above result in the range between 0 to 5.
4. Retrieve Sender IP and Key from sender.
5. After the conversion, Random no is appended to the key and Sender IP in the range between 6 to 10 and in the range between 11 to 15.
6. Calculate the Hash value of the original message and Append the value at the end of the cipher.

V. CONCLUSION

In the previous system is immune against all the known attacks including the adaptive chosen-cipher text attack. In this system, the techniques we have used not only prevent the attacks overcome in the previous systems but also the attacks happen during the transmission also detected and prevented. In the future we are going to develop a better scheme using the same technique to improve the compression efficiency.

REFERENCES

[1] Jiantao Zhou, Oscar C. Au, and Peter Hon-Wah Wong, "Adaptive chosen-ciphertext attack on secure arithmetic coding" *IEEE Trans. Signal Process., vol. 57, no. 5, pp. 1825–1838, May 2009.*

[2] H. Kim, J. T. Wen, and J. D.Villasenor, "Secure arithmetic Coding," *IEEE Trans. Signal Process., vol. 55, no. 5, pp. 2263–2272, May 2007.*

[3] J.T. Wen, H. Kim, and J. D. Villasenor, "Binary arithmetic coding with key-based interval splitting," *IEEE Signal Process. Lett., vol. 13,* pp. 69–72, Feb. 2006.

[4] R. Bose and S. Pathak, "A novel compression and encryption scheme using variable model arithmetic coding and couple chaotic system," *IEEE Trans. Circuits Syst. I, vol. 53, pp. 848–857, Apr. 2006.*

[5] P. W. Moo and X. Wu, "Resynchronization properties of arithmetic coding", in *Proc. IEEE Int. Conf. Image Process., Oct. 1999, pp.* 545–549