SECURE EXP-GOLOMB CODING USING STREAM CIPHER

Jiantao Zhou, Oscar C. Au, and Amanda Yannan Wu

Department of Electrical and Computer Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong, China
Email: {eejtzhou, eeau, amandawu}@ust.hk

ABSTRACT

In this paper, we propose a secure Exp-Golomb coding scheme by incorporating with a stream cipher. Different from the traditional case of using stream cipher where the key stream is directly XORed with the plaintext, we here use the key stream to control the switching between two coding conventions (leading zeros and leading ones). Security analysis results show that the proposed system can provide high level of security with the same coding efficiency and negligible additional cost, compared with a regular Exp-Golomb coding. This scheme could potentially be applied to the state-of-the-art multimedia compression systems, e.g., H. 264, to offer security features.

Index Terms— Exp-Golomb coding, multimedia content protection, cryptanalysis

1. INTRODUCTION

With the widespread use of multimedia content, the multimedia security and digital rights management issues have become increasingly important. A straightforward way to protect multimedia data is to use the traditional encryption methods, e.g., AES and RC4, to encrypt the whole data. Nevertheless, encryption of multimedia files has to be carried out carefully. On one hand, ciphering the complete compressed file may result in excessive computational burden and power consumption at the decoder and perhaps even the server/encoder/transcoder. More importantly, multimedia compressed files typically exhibit well-defined hierarchical structure that could be exploited in several useful ways, e.g., for scalability and transcoding. However, these structures are not recognizable in the ciphertext, and hence, are wasted.

Recently, the integration of encryption and compression has attracted more and more attention in the multimedia encryption community [1, 2, 3, 4, 5, 6, 7]. Wu *et al.* proposed the Multiple Huffman Tree (MHT) scheme by alternately using different Huffman trees in a secret order, without influencing the coding efficiency [1]. However, we showed that this scheme is vulnerable against a chosen-plaintext attack [2]. Grangetto *et al.* devised an efficient encryption scheme by utilizing a randomized arithmetic coding (RAC) [3]. More recently, Kim *et al.* suggested the secure AC (SAC) system, which is an improved version of the interval splitting AC [4, 5]. However, we demonstrated that the SAC could be broken using an adaptive chosen-ciphertext attack with linear complexity [6].

In this paper, we address the problem of designing a lightweight encryption scheme using the Exp-Golomb coding. Different from the Huffman coding, the Exp-Golomb coding conceptually has infinite alphabet size. In addition, the special structure of the Exp-Golomb coding enables efficient encoding and decoding without any table look-up operations. Due to these nice features, the Exp-Golomb coding has been adopted in the state-of-the-art video coding standard such as H. 264. To begin with, we first show the security problem of an existing encryption scheme based on Exp-Golomb coding [7]. We then propose a lightweight encryption scheme utilizing Exp-Golomb coding and a stream cipher. Traditionally, the stream cipher is used in a way that the plaintext is XORed with the key stream. However, in the proposed algorithm, we use the key stream generated by the stream cipher to control the switching between two coding conventions (leading zeros and leading ones). Thanks to the nice property of the stream cipher, we show that high level of security could be provided with the same coding efficiency and low additional computational complexity, compared with a regular Exp-Golomb coding algorithm. In addition, in some practical applications where only ciphertext-only attack is feasible, we can further simplify the system without any security degradation.

The rest of this paper is organized as follows. In section 2, we introduce the Exp-Golomb coding and an encryption scheme based on Exp-Golomb coding. We also show its security problem under ciphertext-only attack. In section 3, we give our proposed system, together with performance analysis. Section 4 presents the security analysis of our proposed system. In section 5, we discuss some practical issues of using the proposed scheme. Section 6 concludes this paper.

This work was supported by the Innovation and Technology Commission of the Hong Kong Special Administrative Region, China (project no. ITS/122/03 and project no. GHP/033/05).

2. SECURITY PROBLEM OF THE SCHEME IN [7]

In [7], Lian *et al.* proposed an Exp-Golomb encryption algorithm (EGEA), which is illustrated in Fig. 1. Each Exp-Golomb codeword is composed of R leading zeros, one '1'-bit separator, and R bits of information Y, where $R \geq 0$, as shown in the second column of Table 1. Let s be an input symbol to be encoded. The encoding procedure of EGEA is performed as follows.

Step 1: Encode s using a regular Exp-Golomb coding. Denote the resulting codeword as $\underbrace{00\cdots0}_{R \text{ bits}}1Y$.

Step 2: XOR Y with a key stream K, namely, $Z = Y \oplus K$.

Step 3: Output the final codeword $\underbrace{00\cdots0}_{R \text{ bits}}1Z$.

For the other symbols to be encoded, similar steps could be conducted to produce the final codewords.

However, it should be noted that it is still not clear how to treat the codeword '1', i.e. the case of $R = 0$. In other words, from [7], we do not know whether or not we should perform XOR operations when we encounter the codeword '1'. In fact, we show in the following that in both cases there are serious problems.

Case 1: Do not perform XOR to the codeword '1'. Note that in the bit stream generated by Exp-Golomb coding, it is not difficult to determine the boundary of different codewords. Since '1's are not XORed with the key stream, we can immediately recover their corresponding symbols. As the codeword length of '1' is 1, we can roughly estimate the probability of the symbols associated with '1' as 2^{-1}. Therefore, in this case, an attacker can recover almost half of the symbols by only observing the ciphertext (ciphertext-only attack).

Case 2: Perform XOR to the codeword '1'. In this case, the codeword '1' may be flipped to '0', depending on the key stream. However, there is a confusion whether a '0' is from the codeword '1' or from the leading zeros. This may lead to the decoding failure. To better demonstrate this, we give a simple example as follows. Let the symbol sequence to be encoded be $a_0a_4a_1$, and the key stream be $101010\cdots$. Hence, the encoded bit stream is 000100010. However, if the symbol sequence to be encoded is $a_{12}a_0a_0$, it is not difficult to verify that the encoded bit stream is also 000100010. Therefore, at the decoder side, it is impossible to distinguish which symbol sequence is the encoded one.

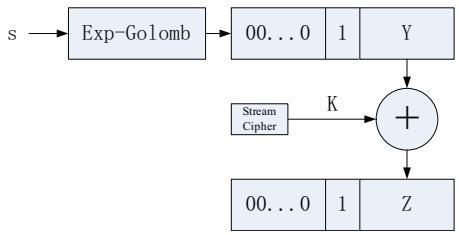

Fig. 1. Schematic diagram of the system proposed in [7].

Table 1. Coding convention 1 and Coding convention 2.

Symbol	Coding convention 1	Coding convention 2
a_0	1	0
a_1	010	100
a_2	011	101
a_3	00100	11000
a_4	00101	11001
a_5	00110	11010
a_6	00111	11011
a_7	0001000	1110000
a_8	0001001	1110001
a_9	0001010	1110010
a_{10}	0001011	1110011
a_{11}	0001100	1110100
a_{12}	0001101	1110101
a_{13}	0001110	1110110
a_{14}	0001111	1110111
$\vdots$	$\vdots$	$\vdots$

3. THE PROPOSED SCHEME

In this section, we present the secure Exp-Golomb coding algorithm, which can be demonstrated in Fig.2. Coding convention 1 and coding convention 2 are shown in Table 1. Specifically, in coding convention 1, leading zeros are used, while in coding convention 2, leading ones are used. The only private information in this system is the seed used in the stream cipher, which is assumed to be 128 bits. Let the symbol sequence to be encoded be $S = s_1s_2\cdots s_N$, and the first N bits of the key stream generated by the stream cipher be $K = k_1k_2\cdots k_N$. The encoding procedure of the proposed secure Exp-Golomb coding is as follows.

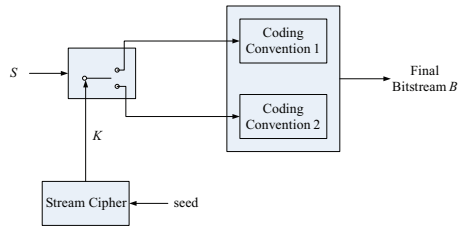

Fig. 2. Schematic diagram of the proposed scheme.

Step 1: Initialize $i = 1$. Fetch the ith symbol from S. If $k_i = 0$, then encode s_i using coding convention 1. Otherwise, use coding convention 2 to encode s_i.

Step 2: Update $i = i+1$. Go to *Step 1* until all the symbols are encoded.

3.1. Performance analysis

It can be easily seen that the generated bit stream has the same length as that generated by a regular Exp-Golomb coding, since in both coding conventions the codeword lengths are the

same for any codes. Therefore, using the secure Exp-Golomb coding, there is no coding efficiency penalty.

Recall that one unique feature of the Exp-Golomb coding is that there is no need to store any coding tables in the encoder and the decoder. The special structure of the Exp-Golomb coding eases the job of the encoding and the decoding significantly. In the secure Exp-Golomb coding, it could be easily found that this property is still preserved. In other words, the encoding and the decoding operations could be performed without the need of any table look-up operations.

In terms of the speed, since we only need to switch between two coding conventions according to a key stream, the encoding and the decoding speed would be almost the same as that of a regular Exp-Golomb coding. In addition, the expected codeword length (ECL) of the Exp-Golomb coding is 3, assuming that the occurrence probabilities of symbols are $2^{-l(a_i)}$, where $l(\cdot)$ denotes the length of the corresponding codeword. Hence, the length of the key stream we need is only 1/3 of that used in a traditional stream cipher, as for each symbol we only need one bit to indicate which coding convention is used. The additional overhead mainly lies on the stream cipher, which in practice could be very efficiently implemented. We will discuss the issue of how to further simplify the complexity of the system under some practical scenarios in section 5.

4. SECURITY ANALYSIS

In this section, we evaluate the security of the proposed secure Exp-Golomb coding scheme. The security analysis is a challenging job for any cryptosystem, since showing robust against known attacks does not preclude the other unknown attacks. Therefore, we evaluate the security under some known attacks, and show that it is secure against these attacks. More specifically, we use the ciphertext-only attack, the chosen-plaintext attack and the chosen-ciphertext attack.

4.1. Ciphertext-only attack

In this attack scenario, the attacker can only have access to the encrypted bit stream. Since the information available to the attacker is very limited, a very commonly used approach is the brute-force attack, whose complexity is related to the key space.

Since the only private information of the proposed system is the seed used in the stream cipher and it is of length 128 bits, then the key space is 2^{128}, which can ensure satisfactory level of security for the digital rights management applications.

Alternatively, the attacker may wish to recover the key stream $K = k_1 k_2 \cdots k_N$ used to control the switching between two coding conventions. Since each k_i is one bit, we can see that the complexity of finding the key stream is 2^N. In order to make this complexity sufficiently large, we impose a constraint on the input sequence length, i.e.,

$$N > 128 \tag{1}$$

which ensures that $2^N > 2^{128}$. Provided that (1) holds, the attacker would rather use the brute-force attack to break the seed used in the stream cipher. Therefore, the key size is sufficiently large to preclude the brute-force attack.

Another large class of ciphertext-only attack is based on the analysis of statistical properties of the final bit stream B. Ideally, the bits in the final bit stream should be i.i.d. Otherwise, the dependence between different bits may provide some information that can be exploited by the attacker to infer the secret key. It is thus important to investigate the statistics of B in order to evaluate the practical security of the proposed system. To this end, we give the following proposition.

Proposition 1: In the final bit stream, $p(B_i = 0) = p(B_i = 1) = 1/2$.

Proof: Let C_1 and C_2 be the events that coding convention 1 and coding convention 2 are used, respectively. We also let

$$\begin{aligned} E_1 &= \{B_i \text{ is from the leading bits}\} \\ E_2 &= \{B_i \text{ is from the separator bit}\} \\ E_3 &= \{B_i \text{ is from the message bits}\} \\ A_j &= \{B_i \text{ is from the codeword } a_j\}, \text{ for } j \geq 1 \end{aligned} \tag{2}$$

We assume that the occurrence probability of a_i is $p(a_i) = 2^{-l(a_i)}$. $p(B_i = 0)$ can then be calculated as follows

$$\begin{aligned} & p(B_i = 0) \\ = & \sum_{m=1}^{2} p(B_i = 0|C_m)p(C_m) \\ = & 1/2 \sum_{m=1}^{2} p(B_i = 0|C_m) \\ = & 1/2 \sum_{m=1}^{2} \sum_{n=1}^{3} p(B_i = 0|C_m, E_n)p(E_n) \\ = & 1/2 \sum_{m=1}^{2} \sum_{n=1}^{3} \sum_{j=1}^{\infty} p(B_i = 0|C_m, E_n, A_j)p(E_n)p(A_j) \\ = & 1/2\Big[p(E_1) + p(E_2) + \sum_{j=1}^{\infty}(2^{-j})p(E_3)\Big] \\ = & 1/2 \end{aligned} \tag{3}$$

This completes the proof.

Hence, the numbers of zeros and ones in the final bit stream B are balanced. In other words, from the first-order statistics, it is difficult to infer the secret key. Then, the proposed system is secure against the ciphertext-only attack.

4.2. Chosen-plaintext attack and chosen-ciphertext attack

In [2], a chosen-plaintext attack was proposed to break the system in [1], with the assumption that the key stream is fixed. However, for a stream cipher, an important property is that the same key stream will not be used more than once. Therefore, even an attacker can somehow find a key stream, it is useless since it will not be used again. This essentially precludes the chosen-plaintext attack, as in [2], and the chosen-ciphertext attack. In section 5, we will elaborate a bit more about how to generate unique key streams for different encryption sessions.

5. IMPLEMENTATION ISSUES

In this section, we discuss some implementation issues on the practical applications of the proposed secure Exp-Golomb coding scheme.

5.1. Some practical issues

Prior to the communications between the encoder and the decoder, a common seed, or called main key (MK), has to be exchanged between the encoder and the decoder. Note that the security of the stream cipher heavily relies on the fact that the same key stream will not be used for more than once. Therefore, if we use solely MK to generate the key stream used in different sessions, the security will be greatly degraded.

In addition, in some video applications, e.g., video surveillance systems, different portions of the video may have different requirements on the level of security. It is desirable to apply different levels of security for different portions of the video.

In order to solve these two problems simultaneously, we can combine the MK, the time stamp (TS), and the slice index (SI) to generate a seed used in the stream cipher by utilizing a hashing function. Namely,

$$Seed = \text{Left}(hash(MK||TS||SI), 128) \tag{4}$$

where $\text{Left}(s, i)$ denotes the first i bits of s; $hash(s)$ is a hashing function, e.g., SHA-1, which generates 160 bits digest; and $s||t$ denotes the concatenation of s and t. Therefore, for different frames in different time, even different slices in one frame, we can use different key streams for encryption. In addition, due to the combination with the time stamp, we can easily realize the random access, which is an important requirement in video surveillance applications.

5.2. How to further simplify the system?

From section 3, we can see that the additional overhead of the proposed system mainly lies on the stream cipher. It should be also noted that the adoption of stream cipher is crucial to resist the chosen-plaintext attack and the chosen-ciphertext attack. However, in some practical scenarios, only the encrypted data is available to the attacker, which essentially precludes the possibility of the chosen-plaintext attack and the chosen-ciphertext attack. In this case, we can further simplify the system by removing the stream cipher. We can exchange a key stream K_1 having length M, which is sufficiently large, between the encoder and the decoder. Then, we obtain a key stream by repeating K_1, i.e., $K' = K_1K_1 \cdots K_1$. Therefore, there is no need of using the stream cipher. Then the complexity of the proposed secure Exp-Golomb algorithm is almost the same as that of a regular Exp-Golomb algorithm.

6. CONCLUSION

In this paper, we have suggested a lightweight encryption scheme based on Exp-Golomb coding and a stream cipher. This algorithm enables us to achieve high level of security with the same coding efficiency and small additional overhead, compared with a regular Exp-Golomb coding. Furthermore, we have shown that in some practical scenarios, the additional overhead could be made negligible without security degradation.

7. REFERENCES

[1] C. Wu and C.-C. J. Kuo "Design of integrated multimedia compression and encryption systems," *IEEE Trans. Multimedia*, vol. 7, pp. 828–839, Oct. 2005.

[2] Jiantao Zhou, Zhiqing Liang, Yan Chen, and Oscar C. Au "Security analysis of multimedia encryption schemes based on multiple Huffman table," *IEEE Signal Proc. Letters*, vol. 14, no. 3, pp. 201–204, March 2007.

[3] M. Grangetto, E. Magli, and G. Olmo "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Trans. Multimedia*, vol. 8, pp. 905–917, Oct. 2006.

[4] J. Wen, H. Kim, and J. Villasenor "Binary arithmetic coding with key-based interval splitting," *IEEE Signal Proc. Letters*, vol. 13, pp. 69–72, Feb. 2006.

[5] H. Kim, J. T. Wen, and J. D. Villasenor "Secure arithmetic coding," *IEEE Trans. Signal Proc.*, vol. 55, pp. 2263–2272, May 2007.

[6] Jiantao Zhou and Oscar C. Au "Adaptive chosen-ciphertext attack on secure arithmetic coding," *Accepted in IEEE Trans. Signal Proc.* 2008.

[7] Shiguo Lian, Zhongxuan Liu, Zhen Ren, and Haila Wang "Secure advanced video coding based on selective encryption algorithms," *IEEE Tran. Consumer Eletronics*, vol. 52, no. 2, pp. 621–629, May 2006.