Cryptanalysis of Some Multimedia Encryption Schemes

G. Jakimoski and K. P. Subbalakshmi

Abstract— **Encryption is one of the fundamental technologies that is used in digital rights management. Unlike ordinary computer applications, multimedia applications generate large amounts of data that has to be processed in real time. So, a number of encryption schemes for multimedia applications have been proposed in recent years.**

We analyze the following proposed methods for multimedia encryption: key-based multiple Huffman tables (MHT), arithmetic coding with key-based interval splitting (KSAC) and randomized arithmetic coding (RAC). Our analysis shows that MHT and KSAC are vulnerable to low complexity known- and/or chosen-plaintext attacks. Although we do not provide any attacks on RAC, we point out some disadvantages of RAC over the classical compress-then-encrypt approach.

Keywords— **multimedia encryption, cryptanalysis, multiple Huffman tables, arithmetic coding, key-based interval splitting, randomized arithmetic coding**

I. Introduction

In the last decades, we have witnessed a rapid growth of networking technologies that provide larger bandwidth and computer technologies that provide greater computational power to the end users. The ease of processing, distributing and storing data on the Internet gave rise to many digital multimedia applications and services. However, the existing wired and wireless IP networks are open networks, and the data transmitted over these networks can be easily copied or modified. So, we have also witnessed the emergence of digital rights management as an important research area for multimedia applications [1]. The goals of the digital rights management technologies include protection of copyrighted multimedia data, authentication, conditional access, etc.

Encryption is one of the major digital rights management enabling technologies. Usually, to provide confidentiality, the data is encrypted using a stream cipher or a block cipher (e.g., DES [2] or AES [3]) in some mode of operation for encryption [4] (e.g., cipher block chaining, output feedback, cipher feedback, output feedback, etc.). However, unlike the ordinary computer applications, multimedia applications generate large amounts of data that has to be processed in real time. Hence, a number of techniques for real-time encryption of multimedia data have been proposed in the past years. Two common approaches to real-time multimedia data encryption are selective encryption [5–19] and entropy coding that provides encryption [20–28].

The design philosophy of selective encryption is to provide faster encryption by encrypting only a small portion of the multimedia data. Without the knowledge of the encrypted data, the adversary will not be able to recover the original data (e.g., image or video). Since traditional schemes can be used for encryption, the issues related to selective encryption are more signal processing than cryptography related. The major issue is to select the important information that will be encrypted, i.e., the information whose encryption will guarantee that the adversary cannot recover useful information about the original image or video.

In the second approach, entropy coding that provides encryption, the entropy coder has two functionalities: compression and encryption. The goal is to improve the efficiency by doing both compression and encryption in a single step. While the traditional entropy coders encode the data in a fixed and public manner, the entropy coders that provide encryption use secret keys to encode the data. The adversary should not be able to decode the data without the secret key. This approach is often combined with selective encryption for greater efficiency.

We analyze the security and the efficiency of MHT (Multiple Huffman Tables), KSAC (Arithmetic Coding with Key-based interval Splitting) and RAC (Randomized Arithmetic Coding). These encryption schemes follow the second approach. That is, the compression and encryption of the data is done in a single step. The encryption in MHT [30] is achieved by using different Huffman tables for different input symbols. The tables as well as the order in which they are applied are kept secret. KSAC [31] is designed to achieve both compression and confidentiality by using keys to specify how the intervals will be partitioned in each iteration of an arithmetic code. The randomized arithmetic coding approach [1] is similar to KSAC. The difference is that the keys are used to specify the order of the intervals instead of the positions where the intervals will be split.

Our analysis shows that both MHT and KSAC are vulnerable to known plaintext attacks. We do not provide any attacks on RAC. However, we point out that the scheme is expected to be less efficient than the standard approach since a generation of one pseudorandom bit per input binary symbol is required.

The paper is organized as follows. Section II provides an overview of the schemes that are analyzed here. Known-plaintext attacks on MHT are presented in Section III. The security and efficiency of KSAC and RAC are discussed in Section IV. The paper ends with concluding remarks.

The authors are with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA, e-mail: gjakimos@stevens.edu, ksubbala@stevens.edu .

This work was supported in part by the National Science Foundation under the grant NSF 0627688.

II. Preliminaries

In this section, we give a brief description of the multimedia encryption schemes analyzed in this paper.

A. Overview of MHT

MHT [30] (Multiple Huffman Tables) is a scheme that performs both compression and encryption by using multiple statistical models (i.e., Huffman coding tables) in the entropy encoder. The secret key which is used for encryption and decryption consists of m distinct Huffman coding tables and an n-tuple $(k_0, \ldots, k_{n-1})$. The m Huffman tables are selected randomly from some public pool of Huffman tables, and the adversary does not know which tables have been selected. The n-tuple $(k_0, \ldots, k_{n-1})$ specifies which table will be used to encode a particular symbol. Namely, the input stream is divided into blocks of n symbols, and the i-th $(0 \leq i < n)$ symbol of each block is encoded using the Huffman table specified by k_i. A high level description of the algorithm is given by the following three steps:

1. Choose m different Huffman coding tables numbered from 0 to $m-1$. The authors suggest $m = 8$.
2. Generate a random vector $(k_0, \ldots, k_{n-1})$, where each k_i is an integer in $\{0, 1, \ldots, m-1\}$. The suggested value for n is 128.
3. Encode the i-th symbol of the data stream using the table specified by $k_{i \bmod n}$.

The public pool of Huffman coding tables is generated from four basic Huffman tables (see Fig. 1) by using Huffman tree mutation (a method introduced in [30]). The first tree is the original Huffman coding tree used to encode the DC coefficients in JPEG. The other trees are obtained by using different training image sets. The Huffman tree mutation process derives a new tree from an old one by swapping the labels of any two branches that stem from a common node. By applying this process, a pool of 2^{14} (2^{12} per basic tree) distinct trees is constructed. In the proposed encoding scheme, the m distinct Huffman tables are randomly and secretly selected from this pool. The codeword lengths for each symbol encoded with the four basic Huffman trees are given in Table I. Clearly, the Huffman trees mutated from a basic tree will have equal codeword lengths as the basic tree. However, a given symbol can be encoded into codewords with different lengths when coding trees mutated from different basic trees are used. This fact is used as a basis for the security of the scheme against known-plaintext attacks. That is, the authors make the following assumption: "for an attacker who does not know the order in which these trees are applied, synchronizing between the symbol stream and the encoded bit stream would be extremely difficult".

B. KSAC and RAC

Arithmetic coding [32] encodes a sequence of symbols with a number (position) in the range $[0, 1)$. It is usually implemented using a recursive procedure. We demonstrate this by the following example that will be used throughout the paper. A subinterval of $[0, 1)$ is associated with each symbol. The length of each subinterval is equal to the probability of the corresponding symbol. Suppose that there are only two possible symbols A and B with probabilities $\Pr[A] = 2/3$ and $\Pr[B] = 1/3$. Then, a possible partitioning would represent A by the range $[0, 2/3)$ and B by the range $[2/3, 1)$. Suppose that the N-symbol sequence to be encoded is $AB\ldots$. The first symbol in the sequence, determines which of the two subintervals will be selected. In our case, the first symbol is A, and we will select the subinterval $[0, 2/3)$. The subinterval $[0, 2/3)$ is partitioned into two subintervals whose lengths are equal to the probabilities of A and B. The second symbol, which is B in our case, determines which of the two subintervals of $[0, 2/3)$ will be selected. This procedure is repeated for all N symbols. The encoding of the sequence is a number

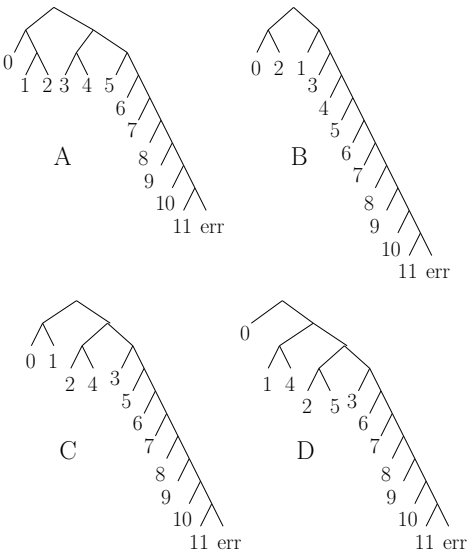

Fig. 1. The four basic trees: (A) the original Huffman coding tree for JPEG DC coefficient coding; (B), (C), (D) Huffman trees trained from three image sets

TABLE I
Total number of different codeword lengths for each symbol when the four Huffman trees of Fig.1 are used together.

Symbol	Tree A	Tree B	Tree C	Tree D	lengths
0	2	2	2	1	2
1	3	2	2	3	2
2	3	2	3	4	3
3	3	3	3	4	2
4	3	4	3	3	2
5	3	5	4	4	3
6	4	6	5	5	3
7	5	7	6	6	3
8	6	8	7	7	3
9	7	9	8	8	3
10	8	10	9	9	3
11	9	11	10	10	3
error	9	11	10	10	3

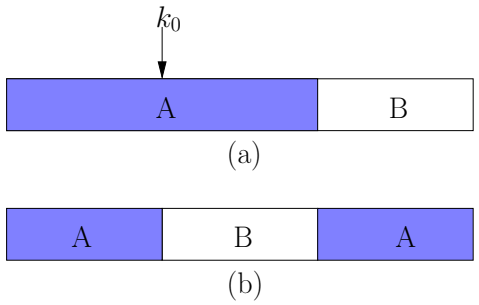

Fig. 2. Key-based interval splitting example: (a) Before split, (b) After split.

in the interval that was selected after N iterations.

The subintervals corresponding to each symbol in the traditional arithmetic coder are continuous and their order is fixed and public. Binary arithmetic coding with key-based interval splitting [31] (KSAC) is designed to achieve both compression and confidentiality by splitting the intervals according to a secret key. So, now we can have more than one interval corresponding to a particular symbol. The sum of the lengths of the intervals corresponding to a particular symbol should be equal to the probability of that symbol. While KSAC keeps the interval(s) corresponding to a given symbol secret by using secret interval splitting, the randomized arithmetic coding (RAC) approach [1] achieves confidentiality by changing the order of the intervals corresponding to the symbols according to a secret key. More details on these schemes are given below.

B.1 KSAC

The concept of key-based interval splitting is depicted in Fig. 2. We use the binary system example that we discussed above. The traditional partitioning of the interval $[0, 1)$ is depicted in Fig. 2(a). KSAC uses a key k_0 to specify where the interval corresponding to A will be split. The interval $[k_0, 2/3)$ is "moved" to the right of the interval corresponding to B, and the final partitioning is shown in Fig. 2(b). The symbol A is represented by two intervals $[0, k_0)$ and $[k_0 + 1/3, 1)$. The symbol B is represented by a single interval $[k_0, k_0 + 1/3)$.

When encoding a sequence of N symbols, one has to select a vector of N keys $k = (k_0, \ldots, k_{N-1})$ that will determine where the interval corresponding to A is going to be split. Fig. 3 depicts the key-based interval splitting for a sequence of two symbols assuming that after the first iteration we have the situation depicted in Fig. 2(b). Fig. 3(a) shows the traditional partitioning and the positions determined by the key k_1 where the intervals will be split. The situation after the key-based splitting is depicted in Fig. 3(b). It is possible that for some key choices a given sequence can be represented by more than two intervals. One such situation is shown in Fig. 3(c). This is not desirable since it further increases the number of bits needed to represent a given sequence. To solve this problem, the authors suggest imposing constraints on the keys [31].

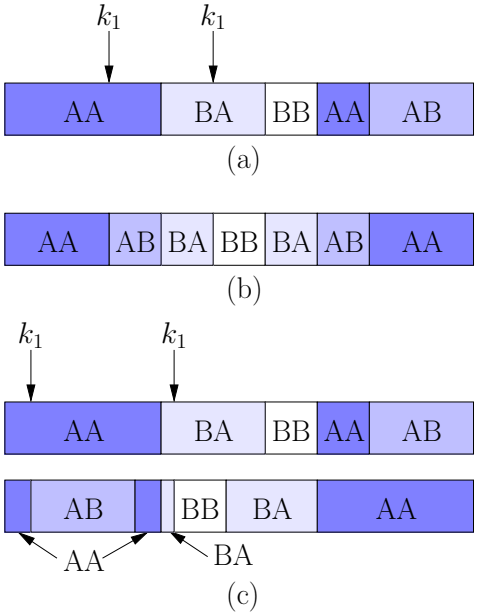

Fig. 3. Interval splitting with two symbols: (a) Traditional partitioning in the second iteration, (b) Key-based partitioning in the second iteration, (c) Inappropriate value for k_1.

B.2 RAC

RAC encryption is based on a random order of the intervals associated with the symbols. This order is secret, and only a synchronized decoder will be able to correctly decode.

The concept of randomized arithmetic coding is depicted in Fig. 4. The traditional arithmetic encoding of the sequence AAB is depicted in Fig. 4(a). The encoding of the same sequence using a randomized arithmetic coder is shown in Fig. 4(b). The ordering of the intervals (i.e., whether the first interval is the interval corresponding to the least probable symbol LPS or the most probable symbol MPS) is determined according to a random and secret bit.

In general, the interval swapping rule when encoding a sequence of N binary symbols $b_0 \ldots b_{N-1}$ is defined as follows. Initialize a pseudorandom bit generator with a random seed S and generate N bits $r_i, i = 0, \ldots, N-1$. The ordering is selected as follows:

```
For i=0:N-1
  If r_i=1 then
    select the order [LPS,MPS] to encode b_i
  else
    select the order [MPS,LPS] to encode b_i
```

The decoder will use the same random seed to regenerate each r_i and correctly decode the sequence of symbols. The pseudorandom bits r_i are generated anew for each new sequence of N symbols.

III. Known-plaintext cryptanalysis of MHT

In this section, we present a known-plaintext attack on MHT (multiple Huffman coding tables) [30]. We consider

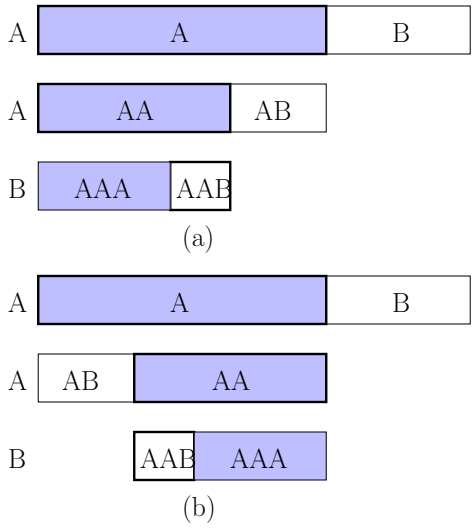

Fig. 4. Illustration of randomized arithmetic coding: (a) Traditional arithmetic coder, (b) Randomized arithmetic coder.

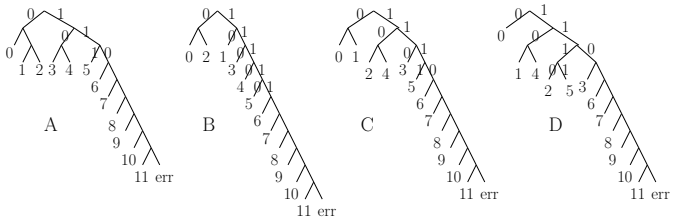

Fig. 5. The labeling of the four basic trees obtained from the encoding of 5.

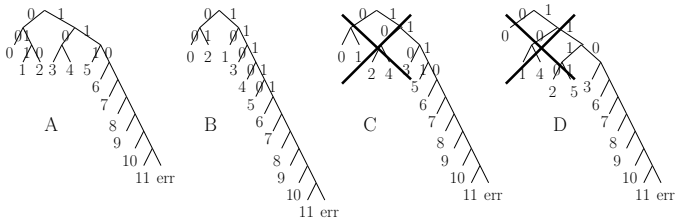

Fig. 6. The labeling of the four basic trees obtained from the encodings of 5 and 2.

two key lifetime possibilities for MHT: long-term keys and per-message keys. In the first case, one key is used to encrypt a large number of messages (plaintexts[1]). In the second case, a new key is generated for each new message. The attack on MHT with long-term keys can be trivially avoided by generating a new (pseudorandom) key for each new message[2]. However, the attack on MHT with per-message keys makes use of the attack on MHT with long-term keys. So, we present both attacks.

A. Cryptanalysis of MHT with long-term keys

In this section, we assume that the adversary has access to many messages and the corresponding ciphertexts that are obtained using the same key[3]. The following example demonstrates how the adversary can recover the secret key.

Suppose that the adversary knows the following three plaintext/ciphertext pairs:

$$\begin{aligned} 53\ldots &\leftarrow 11110101\ldots, \\ 26\ldots &\leftarrow 0101011\ldots, \\ 13\ldots &\leftarrow 10011\ldots. \end{aligned}$$

Knowing the encoding of 5, the adversary labels the edges of the path from the root to symbol 5 in the four basic trees. Then, the adversary checks whether a branch is labeled. If a branch is labeled 1, then he labels the other branch of the pair with 0. If a branch is labeled 0, then he labels the other branch of the pair with 1. The resultant labeling is depicted in Fig. 5.

[1]We adhere to the standard terminology adopted in cryptology and use the words message or plaintext for the data that will be encrypted.

[2]The authors of MHT do not explicitly specify that a fresh key has to be generated for each message. However, we believe that they tacitly make this assumption when stating that the security of the scheme relies on the difficulty of synchronizing the symbol stream and the encoded bit stream.

[3]Our attack works even if the adversary knows only many distinct message headers and the corresponding ciphertexts. He doesn't need to know the complete messages.

Next, using the second message/ciphertext pair, the adversary eliminates the trees C and D since the encoding of 2 begins with zero. This is not possible for the third and the fourth tree. Furthermore, the adversary also labels the branches on the path from the root to symbol 2 in Tree A and Tree B. Finally, he checks whether a branch is newly labeled. If the branch is labeled 1, then he labels the other branch of the pair with 0. If the branch is labeled 0, then he labels the other branch of the pair with 1. The resultant labeling is depicted in Fig. 6.

Finally, using the third message/ciphertext pair, the adversary eliminates the tree A since the encoding of 1 begins with one. This is not possible for the first tree. The final result is depicted in Fig. 7. Note that using only three known message/ciphertext pairs the adversary has determined the correct basic tree and recovered the encoding of the most frequently used symbols.

By using more known plaintext/ciphertext pairs, one can easily recover the Huffman coding table that is used to encode the first symbol of each message block. A general description of the attack is given by the following steps:

1. Construct four unlabeled trees corresponding to the four basic trees that are used in MHT.
2. Given the encoding of the first symbol of a message, label the branches on the path from the root to the particular

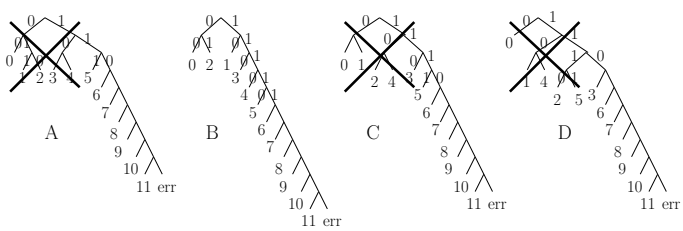

Fig. 7. The labeling of the four basic trees obtained from the encodings of 5, 2 and 1.

symbol in each tree. If some of the branches are already labeled and the existing labeling is not consistent with the current labeling (i.e., symbol encoding), then discard that tree as a possible tree.
3. For each newly labeled branch, label the corresponding branch in the pair.
4. Repeat Step 2 and Step 3 for all known message/ciphertext pairs.

The least probable symbol appears with probability $\approx 2^{-10}$. So, given about 1000 random and known message/ciphertext pairs each symbol will appear at least once on average and the Huffman table for the first symbol can be recovered completely. Once the Huffman table for the first symbol in the message is recovered, we can discard the first symbols of the messages and their encodings from the ciphertexts, and then repeat the same attack to recover the Huffman table used to encode the second symbols of the blocks, and so on. Note that one does not have to recover the complete Huffman table to "attack" the subsequent symbols. It is sufficient to determine the basic tree, and thus determine the codeword lengths.

The attack can be further optimized by using the following techniques :
• Process the messages that start with least frequent symbols first. The encodings of these symbols have largest length and therefore reveal more information about the structure of the Huffman coding tree.
• Use the codeword lengths given in Table I to eliminate the possible basic trees. For example, if the encodings of two messages that start with 5 differ in the fourth bit, then the codeword length of the encoding of 5 is less than 4. This is possible only if the coding tree A is used to encode 5.
• There are only $m = 8$ secret tables per key. Once, we recover these tables, we can just check which one is used to encode the i-th symbols of the blocks.

The complexity of the attack is low, and it can carried out by hand although the key size is quite large $14 \times 8 + 128 \times 3 = 496$ bits.

B. Cryptanalysis of MHT with per-message keys

If the adversary knows where the encodings of the blocks of the plaintext begin within the corresponding ciphertext, then he can find the encoding of each block and apply the attack described in Section III-A by viewing each block as a new plaintext. So, the security of the scheme relies on the assumption that it is hard to find where the encoding of each block begins due to the different codeword lengths when using different tables. However, as demonstrated in this section, this assumption is not true. One can guess which tables are used for the first few (the first four in our case) symbols of each block. Then, he can use the guessed tables to encode the first few symbols of the second block, the third block, etc. If the guess is correct, then the resultant encodings must appear within certain areas of the ciphertext determined by the minimum and the maximum possible codeword lengths. If the guess is wrong, then the probability that the resultant encodings will appear within these areas of the ciphertext becomes negligible when sufficiently many (about 32) known blocks are used. Hence, the adversary can find the right tables for the first few symbols of each block, and then using these tables he can find where the encoding of each block begins.

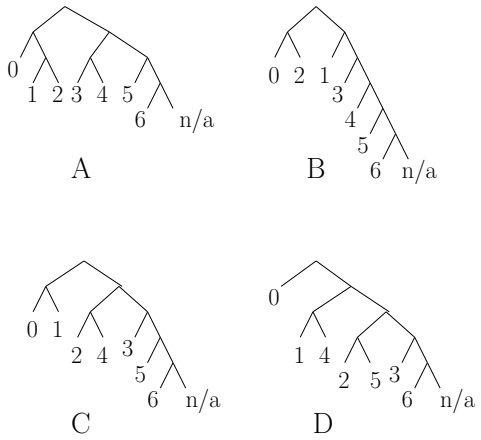

Fig. 8. The pruned variants of the four basic trees. About 97% of the plaintext is encoded using these pruned trees.

Before presenting the details of the attack, let us make the following observations:
• *Given a ciphertext and the encoding of the more frequent symbols (0 – 6), one can easily recover the corresponding plaintext.* Indeed, the probability that a given symbol of a plaintext is one of the symbols 7 – 11 or error is about 1/32. Hence, most of these symbols are "surrounded" by relatively long sequences of the frequent symbols. Since we know the encoding of the frequent symbols, we can easily find the encoding length of the infrequent symbol. Given the length of the symbol and the prefix (we know the encoding of 6), we can easily recover the encoded infrequent symbol.
• *The number of different encodings of the frequent symbols is significantly smaller than the total number of different Huffman tables used in MHT.* Fig. 8 depicts pruned variants of the four basic trees. Each of these trees can be labeled in 2^7 different ways. So, the total number of different encodings of the frequent symbols is 2^9 which is significantly smaller than the number of Huffman tables 2^{14}. The large number of different Huffman tables is due to the large length of the encoding of the least frequent symbols. However, as we previously mentioned, these symbols make only a small portion of the plaintext. So, one does not have to guess their encoding in order to mount an attack.

Suppose now that the adversary knows the first L blocks $B_1 = s_{1,1}|\dots|s_{1,n}$, …, $B_L = s_{L,1}|\dots|s_{L,n}$ of a message M and the ciphertext C corresponding to M. A two-stage attack that recovers the unknown part of the message is described below. The first stage of the attack consists of the following steps:
1. Guess[4] which pruned trees are used to encode the first four symbols of each block.

[4]Guessing a key in cryptanalysis means to try a possible value of a key. In our case, we guess which pruned trees are used to encode the first four symbols of the blocks (e.g., ACAD is a possible guess).

2. Given the encodings of $s_{1,1}, s_{1,2}, s_{1,3}$ and $s_{1,4}$, label the paths from the roots of the four trees assumed in the previous step to the corresponding symbol. Label the branches corresponding to the branches that are already labeled as in Section III-A.
3. Construct a list whose members are 4-tuples of fully labeled pruned trees such that the labels of the trees are consistent with the labels of the partially labeled trees constructed in Step 2. This list gives us the possible encodings of the first four symbols of each block given the encoding of $s_{1,1}|\dots|s_{1,4}$. Since each symbol is encoded using 3 bits on average, this list will have $(2^{7-3})^4 = 2^{13}$ members on average.
4. For each member of the list constructed in the previous step and for each $i = 2, \dots, L$ do the following:

- Find the encoding c_i of $s_{i,1}|\dots|s_{i,4}$ using the current member of the list. If some of the symbols $s_{i,1}, \dots, s_{i,4}$ is one of the less frequent symbols, then encode it as $\underbrace{**\dots*}_{l}$, where l is its length which is known given a pruned tree. In the following, we assume that $*$ matches both 0 and 1.
- Let p_0 be the position of the first bit of the encoding of $s_{i-1,1}|\dots|s_{i-1,4}$. Set $p_s = p_0 + l_m(B_{i-1})$ and $p_e = p_0 + l_M(B_{i-1})$, where $l_m(B_{i-1})$ is the minimal possible length of the encoding of B_{i-1} and $l_M(B_{i-1})$ is the maximal possible length of the encoding of B_{i-1}. Since each symbol has at most three different codeword lengths, the difference $p_e - p_s$ cannot be greater than 256 (assuming $n = 128$).
- Starting at position p_s and ending at position p_e search the ciphertext C for the appearance of the string c_i. If c_i does not appear starting at some position between p_s and p_e, then discard the currently considered member of the list as a possible one and go on to the next member of the list. Otherwise, check the current member of the list for the next value of i.

5. Repeat Steps 1–4 until all possibilities for the four unlabeled pruned trees (guessed in Step 1) are exhausted.

The first stage of the attack has two goals:

- Find the correct (pruned) encoding tables that are used to encode the first four symbols of each block. This is achieved by discarding the incorrect tables using the substring test: the correct encodings c_i must appear in the ciphertext C.
- Find the beginning of the encodings of B_i $(i = 2, \dots, L)$ within the ciphertext C. This is also achieved by the substring test: the position of c_i within C marks the beginning of the encoding of B_i.

Once these two goals are achieved, we can easily find the (pruned) encoding tables used to encode the i-th $(i > 4)$ symbol of each block. For example, we can guess the coding table that is used to encode the fifth symbol. Given the table and the position of the encodings of the fifth symbol within the ciphertext, we can easily check whether the table is correct or not. The probability that a wrong guess will be accepted as right is negligible even for relatively small values of L. Once we find the table used for the fifth symbol of the blocks, we can find the table for the sixth symbol, and so on.

We now roughly estimate the time complexity and the probability of success of the attack. We assume $L = 32$. Let us analyze the first stage of the attack. Note that if a given member of a list constructed in Steps 1–3 is the right one, then it will pass all the substring tests, and it will not be discarded from the list. Let us now estimate the number of wrong guesses (i.e., wrong members of the lists constructed in Steps 1–3) that will pass the first stage. We consider two possible cases:

- A wrong guess for the (pruned) Huffman tables that encode the first four symbols leads to the same encodings of each $s_{i,1}|\dots|s_{i,4}$ $(i = 1, \dots, L)$ as the right guess. This is possible if some of the more frequent symbols does not appear. For instance, let us say that symbol 6 does not appear in $\{s_{i,1}\}_{i=1,\dots,L}$. Then at least two guesses will "survive" the substring tests. Only one of them is correct. However, for $L = 32$, the more frequent symbols will appear at least once on average, and we can uniquely determine the correct (pruned) table with "good" probability. For instance, the probability that 6 will not appear in $\{s_{i,1}\}_{i=1,\dots,L=32}$ is $(1 - 1/32)^{32} \approx 0.36$.

Note that we don't need to know the complete labeling of the trees to execute the second stage. We only need to know which basic pruned trees are used to encode the first four symbols and where each block encoding begins within the ciphertext. So, even if we do not know the encoding of 6 for the first and third symbol of each block, we can still carry out the second stage. Also, we can obtain the encoding of some symbols even if they do not appear. For example, the encoding of 2 in Tree A uniquely determines the encoding of 1, and so on. So, if 1 does not appear, but 2 appears in $\{s_{i,1}\}_{i=1,\dots,L=32}$, we can still obtain the encoding of 1 knowing that Tree A is used to encode the first symbol of each block.

- The second possibility is that a wrong guess encodes the first four symbols of the blocks $B_1, \dots, B_L$ differently than the correct guess, but passes the tests since the incorrect encodings can be found between the starting positions p_s and the ending positions p_e. Suppose that c_i is an incorrect encoding of $s_{i,1}|\dots|s_{i,4}$. Each symbol is encoded using 3 bits on average. Hence, the average length of c_i is 12 bits. If we take into account that some of the symbols $s_{i,1}, \dots, s_{i,4}$ can be one of the less frequent symbols, the average number of known bits of c_i is somewhat less than 12. Hence, we assume that the probability that c_i will match an arbitrary substring of C is 2^{-11}, and the probability that c_i will match any substring of C starting between p_s and p_e is $1 - (1 - 2^{-11})^{256} \approx 2^{-3}$. The probability that this is going to happen for all $i = 2, \dots, 32$ is about 2^{-96}. Since there are 2^{24} possible guesses, a rough estimation of the average number of wrong guesses of the second type that will pass the tests is 2^{-72}.

In summary, after the first stage, we will know with high probability which pruned trees (A, B, C or D) were used to encode the first four symbols of each block. With good probability, we will be able to completely label these pruned trees. We also note that the success probability of the first stage can be significantly improved by using larger L (e.g.,

$L = 64$ or $L = 96$).

Since the beginning of the encoding of each B_i within the ciphertext C was determined in the first stage, our task in the second stage is much easier. The probability that two different trees will encode 32 random symbols into equal bit strings is negligible. So, we can determine with high probability which of the trees A, B, C and D was used to encode the fifth, the sixth, ... , the n-th symbol of each block. Again, with good probability, we will be able to completely label the recovered pruned trees. Now, we can use the information deduced in the attack to recover most of the message M.

The time complexity of the attack is relatively low. Most of the work is done in the first stage. Even a pessimistic estimate assuming that no members of the lists are discarded leads to time complexity of about $2^{24} \times 2^8 \times 2^5 = 2^{37}$. Here, 2^{24} is the average number of different pruned tables that can be used to encode the first four symbols given the encoding of $s_{1,1}|\dots|s_{1,4}$; $2^8 = 256$ is the maximum distance between p_s and p_e; and 2^5 is L.

C. Cryptanalysis of the MHT variants incorporating the proposed security enhancements

The authors have proposed two methods to enhance the security of the basic MHT scheme: selective random bit insertion in the encrypted bit-stream and using different keys for different segments of the plaintext.

As mentioned earlier, the authors assume that MHT is secure against known-plaintext attacks since it is difficult to synchronize the plaintext and the ciphertext due to the different codeword lengths of a given symbol. The selective random bit insertion tries to further increase the difficulty of synchronization by randomly inserting bits at certain positions of the ciphertext. This increases the length of the ciphertext. So, the authors suggest that the bit insertion should not increase the size of the ciphertext more than 1% on average. The effects of this method in the long-term keys case will be negligible. For our per-message key case, this will mean that the maximum distance $p_e - p_s$ is not 256, but 256+2.56. This will only slightly increase the probability that a wrong value c_i will pass the substring test. We need to double the maximum distance $p_e - p_s$ in order to increase this probability twice. This amount of redundancy is unacceptable in practice. Furthermore, one can guess the tables that are used to encrypt the first five or more symbols instead of guessing the tables used for the first four symbols, and mount the same attack. The complexity of the attack will increase 2^5 or more times, but it is still low.

The second proposed method to enhance the security of the scheme is to divide the plaintext into segments and use different keys for each segment. The keys that are used for the different segments are generated using a pseudorandom number generator. The problem with this approach is that we have been able to mount an attack using only 32 known blocks of the plaintext. Furthermore, even with several known blocks, one can deduce a significant amount of information about the encoding tables in use. That means that the segments in this approach should be relatively small (consisting of a few blocks). However, this will significantly increase the time complexity of the scheme since we have to generate tables that will be used to encrypt a few blocks. The complexity of this scheme might be even larger than the traditional encode-then-encrypt approach. In addition, it might be vulnerable to chosen-plaintext attacks while the traditional stream cipher approach is not.

IV. Analysis of KSAC and RAC

In this section, we point out some vulnerabilities of KSAC and compare RAC to the standard approach where one first compresses the data and then encrypts the result using a stream cipher.

A. Attacks on KSAC

In KSAC, the number of keys that specify where to split the intervals in each iteration is equal to the number of symbols in the encoded block. That is, a new key should be generated for each symbol making the scheme very inefficient. To solve this problem, the authors suggest two techniques:

- Reuse of keys. A single key can be used to encode more than one block of N symbols.
- Small keys. The number of positions where the intervals are split should be relatively small. For a binary system, it is suggested each of the keys k_i to be 2 bits long. That is, the number of positions where an interval can be split in a given iteration is four.

We show that these techniques lead to schemes that are vulnerable to known- and chosen-plaintext attacks.

Let us consider the binary system example of Section II-B.1. There are two possible symbols A and B with probabilities 2/3 and 1/3 respectively. Furthermore, assume that there are four possible positions $p_0, \dots, p_3$ where the interval corresponding to A can be split as shown in Fig.9. The adversary knows the values $p_0, \dots, p_3$, but he does not know which one will be selected by the key k_0. Now, suppose that the adversary also knows the encodings $E(S_1), E(S_2)$ of two input blocks S_1 and S_2 such that (i) The first symbol of S_1 is A and its encoding is in the interval $[p_1, p_2)$, and (ii) The first symbol of S_2 is B and its encoding is in the interval $[p_2, p_3)$. Clearly, this can happen only if $k_0 = p_2$. So, given the previous knowledge, the adversary can easily recover k_0. Let us estimate the probability p_s that one can find at least two such blocks given x blocks $S_1, \dots, S_x$ selected randomly according to the source probability distribution and their encodings under the same key. We assume that the distance between any two consecutive positions p_i, p_{i+1} is $\frac{1}{5} \cdot \frac{2}{3}$ and that the encodings are uniformly distributed in the interval $[0, 1)$. The probability that no two out of the x encodings satisfy the aforementioned properties is equal to the probability that the x encodings belong to $[0, 1) - [p_1, p_2)$ or $[0, 1) - [p_2, p_3)$, i.e. at most $2(1 - \frac{2}{3}\frac{1}{5})^x$. The probability that at least one pair S_i, S_j satisfies the aforementioned properties is

$$p_s \geq 1 - 2\left(1 - \frac{2}{3}\frac{1}{5}\right)^x.$$

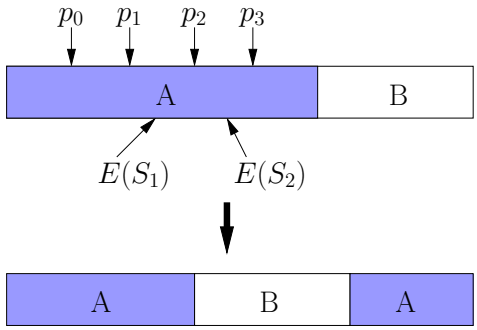

Fig. 9. Recovering k_0. S_1 begins with A and its encoding is in the interval $[p_1, p_2)$. S_2 begins with B and its encoding is in the interval $[p_2, p_3)$. Hence, k_0 must be equal to p_2.

So, given 10 known plaintexts encoded using the same key, the adversary can recover k_0 with probability $p_s \gtrapprox 0.5$, and given 20 known plaintexts, the adversary can recover k_0 with probability $p_s \gtrapprox 0.9$.

The previously described procedure can be applied in each iteration to recover the keys $k_1, \ldots, k_N$. We simplify our analysis and assume that the distance between two consecutive positions where a split can occur decreases by 2/3 in each iteration, and that the probability of success in the n-th iteration is

$$p_s = 1 - 2\left(1 - \left(\frac{2}{3}\right)^n \frac{1}{5}\right)^x.$$

The number of required known plaintext to achieve a given probability of success in the n-th iteration is

$$x = \frac{\ln \frac{(1-p_s)}{2}}{\ln\left(1 - \left(\frac{2}{3}\right)^n \frac{1}{5}\right)}.$$

According to the previous analysis, the number of required known plaintexts to achieve a given success probability increases rapidly with the number of iterations. So, one might suggest using large blocks to counter this kinds of attacks. However, there are several pitfalls in this approach:

- Given $k_0, \ldots, k_{n-1}$ and the encoding of any unknown block $s_0|\ldots|s_{N-1}$, the adversary can recover the first n symbols of the block. So, one does not have to recover all the keys in order to recover some percentage of the plaintext.
- The known-plaintext attack can be used in combination with other attacks to recover the keys. For example, one can find 10 out of the N keys using our method, and then use exhaustive search to find the rest of the keys.
- The scheme will not be secure according to the definitions used in cryptography. These definitions do not allow recovery of even one unknown bit of the plaintext given the ciphertext and a part of the plaintext. This is not the case here. With a few known plaintext/ciphertext pairs we can recover k_0, and using k_0 we can recover the first symbol of any other message (block).
- We can significantly improve the attack by using chosen plaintexts. The large number of plaintexts that we need in the known-plaintext attack is due to the exponential decrease of the selected intervals with each iteration. Since the considered intervals are very small we need a large number of known plaintexts so that at least few of the encodings fall in the considered interval. This problem can be resolved by using chosen plaintexts. Namely, using the previously recovered keys $k_0, \ldots, k_{n-1}$, we can choose x plaintexts whose encodings will belong to the considered interval with probability 1. Now, to achieve a given success probability p_s in each iteration we need a fixed number of chosen plaintext/ciphertext pairs. Applied to our binary system example, a chosen-plaintext attack will recover a key of length $N = 1000$ with 20% probability using $x = 50$ chosen plaintexts in each step or 50000 chosen plaintexts in total.

B. Comments on the efficiency and security of RAC

Probably aware of the possible vulnerabilities when reusing a key, the authors of RAC suggest using different keys (i.e., different values of the (pseudo)random bits r_i) for different input blocks. The keys used for the different blocks are generated using a pseudorandom bit generator (PRBG). This approach is similar to the standard stream cipher approach where one first compresses the input data and then encrypts the result by XOR-ing it with the output of a pseudorandom bit generator. Although there are no apparent attacks on RAC, there are some disadvantages of RAC over the standard approach:

- In the standard approach, one needs to generate one pseudorandom bit per bit of the compressed data. In RAC, one has to generate one pseudorandom bit per input binary symbol. Since, it is expected that the input data is longer than the compressed data, the RAC efficiency is expected to be worse than that of the standard approach.
- Provably secure schemes are more appreciated than heuristic scheme. The standard approach is proven secure assuming that the PRBG is cryptographically secure. If one can break the encryption scheme, then one can distinguish the output of the PRBG from a random sequence. So far, such proof has not been provided for RAC.

V. Conclusion

We have analyzed the security and efficiency of some recently proposed schemes for multimedia encryption: key-based multiple Huffman tables (MHT), arithmetic coding with key-based interval splitting (KSAC) and randomized arithmetic coding (RAC). We showed that MHT and KSAC are vulnerable to known-plaintext attacks. We also compared RAC to the classical compress-then-encrypt approach, and argued that there are no advantages of using RAC over the standard approach in terms of efficiency and security.

References

[1] M. Grangetto, E. Magli and G. Olmo, Multimedia selective encryption by means of randomized arithmetic coding, IEEE Transactions on Multimedia, vol. 8, no. 5, 2006.
[2] Data Encryption Standard, FIPS PUBS 46-3, 1999.
[3] Advanced Encryption Standard, FIPS PUBS 197, 2001.

[4] Recommendation for Block Cipher Modes of Operation, NIST Special Publication 800-38A, 2001.

[5] J. Meyer and F. Gadegast, Security Mechanisms for Multimedia Data with the Example MPEG-1 Video, Project Description of SECMPEG, Technical University of Berlin, Germany, May 1995.

[6] G. A. Spanos and T. B. Maples, Performance Study of a Selective Encryption Scheme for the Security of Networked, Real-time Video, Proceedings of 4th International Conference on Computer Communications and Networks, Las Vegas, NV, September 20-23, 1995.

[7] L. Tang, Methods for Encrypting and Decrypting MPEG Video Data Efficiently, Proceedings of the 4th ACM International Multimedia Conference, Boston, MA, November 18-22, 1996, pp. 219–230.

[8] L. Qiao and K. Nahrstedt, A New Algorithm for MPEG Video Encryption, Proceedings of the 1st International Conference on Imaging Science, Systems and Technology (CISST 97), Las Vegas, NV, July 1997, pp. 21–29.

[9] C. Shi and B. Bhargava, A Fast MPEG Video Encryption Algorithm, Proceedings of the 6th International Multimedia Conference, Bristol, UK, September 12-16, 1998.

[10] A. M. Alattar, G. I. Al-Regib and S. A. Al-Semari, Improved Selective Encryption techniques for Secure Transmission of MPEG Video Bit-Streams, Proceedings of the 1999 International Conference on Image Processing (ICIP '99), vol. 4, pp. 256–260, Kobe, Japan, October 24-28, 1999.

[11] C. Shi, S.-Y. Wang and B. Bhargava, MPEG Video Encryption in Real-Time Using Secret key Cryptography, 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, NV, June 28 - July 1, 1999.

[12] H. Cheng and X. Li, Partial Encryption of Compressed Images and Video, IEEE Transactions on Signal Processing, vol. 48, no. 8, pp. 2439–2451, 2000.

[13] M. Van Droogenbroeck and R. Benedett, Techniques for a Selective Encryption of Uncompressed and Compressed Images, Proceedings of Advanced Concepts for Intelligent Vision Systems (ACIVS) 2002, Ghent, Belgium, September 9-11, 2002.

[14] A. Servetti and J. C. De Martin, Perception-based partial encryption of compressed speech, IEEE Trans. Speech Audio Process., vol. 10, no. 8, pp. 637-643, 2002.

[15] M. Podesser, H.-P. Schmidt and A. Uhl, Selective Bitplane Encryption for Secure Transmission of Image Data in Mobile Environments, 5th Nordic Signal Processing Symposium, Norway, October 4-7, 2002.

[16] A. Pommer and A. Uhl, Selective Encryption of Wavelet-Packet Encoded Image Data, Multimedia Systems Journal, vol.9, no.3, pp. 279–287, 2003.

[17] W. Zeng and S. Lei, Efficient Frequency Domain Selective Scrambling of Digital Video, IEEE Transactions on Multimedia, vol. 5, no. 1, pp. 118–129, 2003.

[18] D. Kundur and K. Karthik, Video fingerprinting and encryption principles for digital rights management, Proc. IEEE, vol. 92, no. 6, pp. 918-932, 2004.

[19] T. Lookabaough and D. C. Sicker, Selective encryption for consumer applications, IEEE Commun. Mag., pp. 124-129, 2004.

[20] D. Jones, Applications of splay trees to data compression, Commun. ACM, pp. 996-1007, 1988.

[21] H. A. Bergen and J. M. Hogan, Data security in a fixed-model arithmetic coding compression algorithm, Comput. Security, vol. 11, 1992.

[22] H. A. Bergen and J. M. Hogan, A chosen plaintext attack on an adaptive arithmetic coding compression algorithm, Comput. Security, vol. 12, pp. 157-167, 1993.

[23] J. G. Clearly, S. A. Irvine and I. Rinsma-Melchert, On the insecurity of arithmetic coding, Comput. Security, vol. 14, pp. 167-180, 1995.

[24] X. Liu, P. G. Farrell, and C. A. Boyd, Resisting the Bergen-Hogan attack on adaptive arithmetic coding, in Proc. 6th IMA Int. Conf. Cryptography and Coding, 1997.

[25] P. W. Moo and X. Wu, Resynchronization properties of arithmetic coding, in Proc. IEEE Int. Conf. Image Processing, 1999.

[26] P. W. Moo and X. Wu, Joint image/video compression and encryption via high-order conditional entropy coding of wavelet coefficients, in Proc. IEEE Int. Conf. on Multimedia Computing and Systems, 1999.

[27] C.-P. Wu and C.-C. J. Kuo, Fast Encryption Methods for Audiovisual Data Confidentiality, SPIE International Symposia on Information Technologies 2000, Boston, MA, November 2000, pp. 284–295.

[28] C.-P. Wu and C.-C. J. Kuo, Efficient Multimedia Encryption via Entropy Codec Design, Proceedings of SPIE Security and Watermarking of Multimedia Content III, vol. 4314, San Jose, CA, January 2001.

[29] M. Grangetto, A. Grosso, and E. Magli, Selective encryption of JPEG 2000 images by means of randomized arithmetic coding, in Proc. IEEE Int. Workshop on Multimedia Signal Processing, 2004.

[30] C. -P. Wu and C. -C. Jay Kuo, Design of integrated multimedia compression and encryption systems, IEEE Transactions on Multimedia, vol. 7, no. 5, pp. 828–839, 2005.

[31] J. 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.

[32] G. Langdon and J. Rissanen, Compression of black-white images with arithmetic coding, IEEE Trans. Commun., vol. COM-29, no. 6, pp. 858–867, 1981.