

## Fast Self-synchronous Content Scrambling by Spatially Shuffling Codewords of Compressed Bitstreams

Wenjun Zeng, Jiangtao Wen and Mike Severa  
PacketVideo Corporation, San Diego, CA 92121

### ABSTRACT

This paper presents a content access control method by spatially shuffling codewords of the compressed bitstream. This approach is lightweight and incurs no bit overhead. One important advantage of the approach is that the resulting scrambled bitstream can be made compliant to the compression format, thus providing some level of scalability, error resiliency, network friendliness and capability of performing signal processing directly on the encrypted bitstream. In addition, we propose a method for generating or updating the shuffling tables on the fly, based on encrypting some local-content-specific bits using a standard cipher. This local-content-specific bits based table generation process is self-synchronous, which is critical in the presence of packet loss. It also enhances the resistance of this spatial shuffling approach to plain-text attack

### 1. INTRODUCTION

Access control is a central piece of an intellectual property protection system. One common method for access control is through encryption. It has been recognized [4][5][6] that, for many applications, it is required that content encryption provides error resiliency, scalability, network friendliness and capability of performing signal processing *directly* on the encrypted bitstream, just as is required for unencrypted content formats. In addition, for some applications with real-time constraint, such as wireless multimedia streaming to low-power devices or digital cinema applications, a particularly desirable feature is low processing overhead, which can usually be achieved through selective encryption and/or some lightweight encryption techniques. A “wholesale” encryption of the entire content bitstream is usually not desirable or even feasible.

The basic concept of selective encryption is to encrypt only a subset of the content bitstream to be protected. When done properly, the resultant cipher text can still be compliant, to certain level, to the content’s original compression syntax, which inherently provides a unified solution addressing the aforementioned requirements [5][6]. Syntax compliance inherits network friendliness, error resilience as well as the possibility of performing various types of signal processing on the encrypted bitstream directly. In addition, the encrypted bitstream, without decryption and re-encryption, will work nicely with protocols designed for the transport of standards-based compressed bitstreams. There will be no marker emulation problem [5], as would be the case for wholesale direct encryption. This is especially attractive for transmitting pre-encrypted (as opposed to encryption on the fly) content using standard transport protocols.

Selective encryption/scrambling schemes for the MPEG compressed video data have been proposed in the literature. Depending on how significant the impact of the selected subsets of the bitstream on the visual quality, and on how predictable/recoverable the selected subsets are based on other unencrypted data, the resultant encrypted bitstreams may have different levels of security. Selective encryption may also incur coding bit overhead, which is another factor to be considered when designing a selective encryption scheme. Some of the previously proposed schemes, e.g., [2][3][4][5], could result in an encrypted bitstream that is still format compliant. For example, Tang [2] used random permutation order, as opposed to the normal zigzag order, for run-length coding. The scheme is very lightweight and is fully format compliant, but it may incur a coding bit overhead of up to over 50%, and is vulnerable to plain-text attack, as well as cipher-text attack (by making use of frequency statistics) [7]. Zeng et. al. [4] proposed a selective scrambling scheme where MVs and a selected subset of DCT coefficients are spatially shuffled in the transform domain, prior to entropy coding. This scheme is fully format-compliant, but incurs some bit overhead. We proposed a framework that could achieve syntax compliance in any format/standard through a unique compliance-preserving encryption method of variable length coded fields in compressed bitstreams in [5]. This method, however, will generally incur bit overhead. There have also been some layered approaches [1][8] that do not result in a format-compliant encrypted bitstream. Some additional header overhead is usually incurred for such approaches.

In general, any encryption applied prior to entropy coding may result in format-compliance, but with the expense of bit overhead. In this paper, we present a new spatial shuffling approach that works directly in the *compressed* domain. This approach is lightweight, format-compliant, thus providing some level of scalability, error resiliency, network friendliness and capability of performing signal processing *directly* on the encrypted bitstream. It is secure and incurs no bit overhead. This approach is part of the selective encryption framework we proposed that has been recently adopted in MPEG4 IPMP (intellectual property management and protection) extensions standard [11]. A self-synchronous local-content-specific bits based shuffling table generation process is also proposed to combat packet loss and plaintext attack. The proposed method is especially suitable for packet-based streaming applications.

### 2. SPATIAL SHUFFLING OF CODEWORDS IN COMPRESSED BITSTREAMS

The basic idea of this approach is to spatially shuffle codewords of the compressed bitstream in a way such that the resultant

bitstream complies with the compression format as much as possible, and is secure from attacks. Since it is simply a reorganization of codewords within a compressed bitstream, with the structure information (header/marker, etc.) intact, no bit overhead is incurred, as opposed to the approach of shuffling transform coefficients proposed in [4].

To better illustrate the idea, we use MPEG-4 video as an example. In the following discussions, we will refer to a collection of codewords that are to be shuffled together as a *basic shuffling unit*. For example, a basic shuffling unit could be a single VLC or FLC codeword, or a code-stream corresponding to an 8x8 block. We will divide the bitstream into groups of basic shuffling units, where each group may have a different number of basic shuffling units and the basic shuffling units of different group are to be shuffled separately. An exhaustive search based attack needs to perform  $S!$  trials to figure out the shuffling table, where  $S$  is the size of the shuffling table [4]. The spatial boundary for each group (referred to as *clear-text unit*) could be a video packet boundary, a few rows of MBs, or a frame. The followings are some special cases that warrant some discussion.

MB as basic shuffling unit: in this case, the bitstream for the information corresponding to one MB will be shuffled as a basic unit. Since an MB bitstream is a self-contained unit, the resultant scrambled bitstream will be fully compliant to the MPEG4 format.

Coded 8x8 block as basic shuffling unit: in this case, a coded 8x8 block bitstream will be used as a basic shuffling unit. No MB level information such as MV and MCBPC/CBPY (variable length codes that are used to derive the macroblock type and the coded block pattern) will be involved. Again, since each coded 8x8 block bitstream is a self-contained unit, the resultant scrambled bitstream will be fully format compliant. However, because INTRA and INTER DCT information are coded with different VLC tables, INTRA block bitstreams and INTER-block bitstreams should be shuffled separately, if full format compliance is desired. If all 8x8 blocks, either coded or not coded, of coded MBs are subject to shuffling, then the MCBPC/CBPY needs to be modified accordingly to make sure the resultant bitstream is format compliant. Note that this modification will have negligible impact on the bit rate. Note also that since intra DCs are often coded with codes of known length, they can be encrypted using common encryption algorithms [2][5], instead of being involved in the shuffling process, i.e., the intra-DC can stay in its original block position while all other run-level codewords of a block will be shuffled.

Run-level codeword as basic shuffling unit: the run-level codewords within a group will be shuffled amongst each other. Let us denote the coded bitstream for an 8x8 block as  $\{DC$  (for intra block only),  $RL_0, RL_1, \dots, RL_i, \dots, RL_{last}\}$ , where  $RL_i$  is the run-level codeword associated with the  $i$ -th non-zero coefficient in the block. We can group run-level codewords from different 8x8 blocks together according to their codeword index. For example, any run-level codewords whose codeword index is in the range of  $[0, T_1)$  will be grouped together and shuffled using one shuffling table. Similarly, any run-level codewords whose codeword index is in the range of  $[T_1, T_2)$  will be grouped together and shuffled using another shuffling table, and so on.

Two special cases are grouping based on the index range of  $[0, 63]$ , and a series of grouping based on the index ranges  $[0,0]$ ,  $[1,1]$ ,  $[2,2]$ , ...,  $[T, T]$ . The former case will result in the largest shuffling space, while the latter case results in the smallest shuffling space for each individual group.

It should be pointed out that different 8x8 blocks usually have different numbers of run-level codewords. The number of run-level codewords for each block should remain the same before and after shuffling in order to allow de-shuffling to work. This is achieved by making sure that the "last" field is preserved. When shuffling a coefficient that is in the last position in a block, and thus has its "last" field set to '1', it is possible that that coefficient is shuffled into the same block or some other block where it is no longer in the last position. In that case the "last" field must be set to '0'. Conversely, it is possible that some other coefficient that was originally not in the last position gets shuffled to the same or some other block and ends up in the last position in that block. In that case the "last" field must be set to '1'. Since the "last" field is not separable from the rest of the codeword, changing its value will require changing the entire codeword, which on average will introduce a slight overhead. Another way to deal with this issue is to never shuffle the last run-level codeword in a block.

The sign bits of the run-level codewords are not necessarily to be shuffled, i.e., all the sign bits can stay in their original positions, and can be subject to encryption instead of shuffling.

Since different frames/MBs have different levels of importance to visual quality, in order to reduce complexity the spatial shuffling can be applied only to selected portions of the bitstream such as I frames, Intra-coded blocks, and a selected subset of the run-level codeword index.

For run-level codeword based shuffling, a few interesting observations can be made: 1) the scrambled bitstream resembles an MPEG bitstream. As a result, most operations that work on MPEG bitstreams, e.g., transport packetization, frame dropping etc, can still work nicely on the scrambled bitstream. However, without de-scrambling, the number or position of decoded coefficients for some 8x8 blocks may exceed 64. An "error-resilient" decoder should be able to handle this situation (without crash), though, since it knows where an 8x8 block bitstream ends based on the "last" flag. Note that it is possible to generate a fully format-compliant scrambled bitstream by truncating some of the last un-shuffled run-level codewords, albeit at the cost of some video quality degradation; 2) Some flexibility for compressed-domain signal processing without deshuffling may be lost, due to the loss/uncertainty of the actual frequency/spatial location of each decoded coefficient. For example, requantization and watermarking are still doable in the transformed domain, but may be adversely affected to certain degree; and 3) synchronization in the case of channel error. Since the number of run-level codewords for each 8x8 block bitstream remains unchanged after scrambling, when there is NO bit error in the scrambled bitstream this information will be available to generate the shuffling table size needed for de-scrambling. However, if there are some channel errors, the number of run-level codewords for some 8x8 blocks may be erroneous, thus de-scrambling would be impossible. For packet-based network, one way to reduce this adverse effect is to shuffle only within a video packet so that if a packet is lost, error concealment will be applied, as opposed to de-scrambling.

### 3. ENCRYPTION-BASED SELF-SYNCHRONOUS SHUFFLING TABLE GENERATION

Content access control by shuffling is usually subject to plaintext attack, where the attacker can reverse engineer the shuffling table if he has access to both the plaintext and cipher text. Therefore, it is important to be able to change the shuffling table over time, e.g., from one shuffling group to another.

One way to achieve this is to constantly change the key used in generating the shuffling table. This may pose significant burden on the key management system, especially in the presence of packet loss. A better approach, as described here, is to generate the shuffling table based on encrypting some “local” information that is not involved in the shuffling. Shuffling based on tables generated this way is self-synchronous, which is highly desirable in the presence of packet loss.

To this end, shuffling tables are updated based on the cipher text output of applying a standard cipher (e.g., DES) to some bits specific to the local (spatially as well as temporally) information. These local-content-specific bits could be any bits in the local bitstream that are NOT going to be involved in the shuffling process (therefore are available prior to de-shuffling). For example, if we want to secure INTRA and INTER MB DCT RUN/LEVEL information through shuffling, then, a good candidate for the “local bits” that can be used in generating the shuffling table is the DCT signs, as they are easily available, usually very randomly distributed, and not to be involved in shuffling (could be encrypted instead). When MB or block is to be used as a basic shuffling unit, other local information bits such as the time stamps can be used. Since the local-content-specific bits are local and varying, the resultant shuffling tables will be updated as well. The shuffling tables are resistant to plain-text attack, as long as the standard cipher used in the table generation process is resistant to plain-text attack.

The following is a very simple approach that can be used as an example to illustrate the idea of encryption based self-synchronous shuffling table generation. Here again, we use MPEG-4 video as an example, and assume that certain fixed length coded fields, such as DCT signs and INTRA DC information are encrypted using DES (in ECB mode which is self-synchronous) with key  $K_F$ , but not shuffled, while Run-Level codewords are shuffled based on a shuffling table generated using the following process.

- a. We use some ciphers such as SEAL [10] (based on a less frequently changed key  $K_L$ , e.g., a fixed key for the whole video) to generate a random bit sequence  $R_L$  of sufficient length  $L$  ( $L$  should be larger than any  $bit\_length * K$  where  $K$  is the number of codewords in a shuffling group, and  $bit\_length$  is the smallest integer that is no less than  $\log_2(K)$ ). The same  $R_L$  will be repeatedly used for all shuffling groups within the lifetime of the key  $K_L$ . As will be discussed later,  $R_L$  will serve as a “master” random sequence to make sure the space of the shuffling tables generated in Step  $d$  is sufficiently large.
- b. Then for each shuffling group with  $K$  RUN-LEVEL codewords to be shuffled, collect the  $K$  key- $K_F$ -encrypted sign bits of the codewords. Denote the concatenation of these  $K$  sign bits as  $R'$ , we will encrypt  $R'$  again using DES (in ECB mode) with a dedicated key  $K_T$ , and denote

the output as  $R$ . We then repeat  $R$   $bit\_length$  times to get  $Rr$  (i.e.,  $Rr=RRRRR\dots R$ ). The repetition also limits the computation to only encrypting  $K$  bits. Note that  $R$  is only used to generate the shuffling table. It is *never* in the clear without knowing the key  $K_T$ , as opposed to  $R'$ .

- c. XOR  $R_L$  and  $Rr$ , the output is denoted as  $Rc$
- d. Divide  $Rc$  into  $K$  non-overlapped segments, each with  $bit\_length$  bits. Create a shuffling table where each input index value  $i$  (from 0 to  $K-1$ ) is mapped to an output index value determined by the  $i$ th segment of  $Rc$ 
  - i. In the case that  $K$  is not a power of 2, the output index value will be converted to that index value modulo  $K$  so that it lies in the range  $[0, K-1]$ .
  - ii. In case two or more input index values are mapped to the same output index value, we will re-assign, in a pre-determined order, un-used output index values (in the range from 0 to  $K-1$ ) to those conflicting input index values.

**Security and Complexity:** Note that, given a fixed  $R_L$ , the possible number of shuffling tables is only  $2^K$  (determined by  $R$  of length  $K$ ). However, the attacker does not know which subset (of size  $2^K$ ) of the superset of  $2^{bit\_length * K}$  tables is used, because  $R_L$ , with a length of  $bit\_length * K$  bits, is unknown. As a result, for exhaustive search attack, the number of trials the attacker needs to conduct is the lesser of  $2^{bit\_length * K}$  (approximately equals  $K^k$ ) and  $K!$  [4], which is always  $K!$ . The scheme is also resistant to plaintext attack, since  $R$  is generated based on DES encryption which is resistant to plaintext attack, and  $R$  is never in the clear without knowing the key  $K_T$  and  $K_L$  (even though the attacker may know some plaintext, i.e., both  $R'$  and  $Rc$ ). The complexity involved in this shuffling table generation process is encryption of  $K$  bits, which is much less than straightforward encryption of  $K$  run-level codewords.

### 4. SIMULATION RESULTS

Although the general idea works for any compression formats, in our simulations, we focused on MPEG-4 video error resilient mode with data partitioning. In this mode, the MB coding type information and motion vector information is separated from the texture information by motion marker for each video packet. The texture information is VLC-coded, with a 1-bit “sign” field present in all run-level codewords. Video packets are delimited by a byte-aligned 17-bit resynchronization marker, and a fixed-length index-to-first-MB information field is put at the beginning of each video packet to provide additional error recovery and error detection capability.

We tested the spatial shuffling idea using each video packet as the *clear-text unit* for shuffling. This configuration provides good error resiliency, but with the compromise of lower level of security. When some video packets only contain a small number of MBs, the shuffling space may not be sufficiently large. As a result, the scrambling effect and the resistance to attack may be limited. This is especially true when only MB or block is used as the basic shuffling unit, as shown in the BottomLeft of Fig. 1 where some local structure can be somewhat guessed. In this case, since QP is relatively small, a video packet (with a pre-

determined total number of bits) in many cases contains only a few MBs. As a result, the shuffling is restricted to local small regions, and the scrambled image is thus somewhat guessable.

Shuffling run-level codewords will increase the shuffling space and further mess up the visual quality significantly. This can be seen from the BottomRight of Fig. 1 where the first 10 run-level codewords in each block are also shuffled (i.e., codewords in the index ranges of [0,0], [1,1], [2,2], ..., [9, 9] are grouped and shuffled separately. DCT sign bits are encrypted instead of being shuffled, and are used for generating the shuffling tables). To further increase the scrambling effect and the resistance to attack, a larger clear-text unit such as a pre-determined number of rows of MBs should be used. Run-level codewords in different index ranges can also be mixed together to further increase the shuffling space. It is worthwhile to point out that both scrambled bitstreams in Fig. 1 are rendered using PV MPEG-4 player, without crash.

Fig. 2 shows that the simple attack by setting DC of luminance component to 128, DC of chrominance components to 0, and all MV to 0 will not work for the proposed scheme. This attack simply reduces the visual scrambling effect of shuffling DC and MV information by setting the false MV and DC to most likely values. One potential attack is correlation-based attack where spatial/temporal smoothness constraint can be used to search for a good estimate of the video frame. This attack, however, is generally difficult to construct due to the diversity of high-level object structure and the uncorrelated nature of the coefficient image, particularly when there is no prior knowledge about the content of the video. Some fields such as intra-coded DC and MV are more vulnerable to such attack. To further enhance the security, these fields can be encrypted [5], instead of being shuffled.

## 5. CONCLUSION

A content access control method by spatially shuffling codewords of the compressed bitstream is proposed. A self-synchronous local-content-specific bits based shuffling table generation process is also proposed to increase the resistance to packet loss and intentional attacks. This approach offers a number of advantages, especially suitable for low-power devices in an error prone network.

## 6. REFERENCES

[1] J. Meyer and F. Gadegast, "Security Mechanisms for Multimedia-Data with the Example MPEG-1-Video," *Project description of SEC MPEG*, Technical University of Berlin, Germany, 5/95.

[2] L. Tang, "Methods for Encrypting and Decrypting MPEG Video Data Efficiently," *Proc. of ACM Multimedia '96*, pp. 219-230, Boston, MA, 11/96.

[3] C. Shi and B. Bhargava, "A fast MPEG video encryption algorithm," *Proc. ACM Multimedia '98*, pp. 81-88, 1998.

[4] W. Zeng and S. Lei, "Efficient frequency domain video scrambling for content access control," *Proc. ACM Multimedia '99*, pp. 285-294, Nov. 1999. An extended version also to appear in *IEEE Trans. Multimedia*, 2002.

[5] J. Wen, M. Severa, W. Zeng, M. Luttrell and W. Jin, "A format compliant configurable encryption framework for access control of multimedia," in *IEEE Workshop on Multimedia Signal Processing*, pp. 435-440, Oct. 2001.

[6] J. Wen, M. Severa, W. Zeng, M. Luttrell and W. Jin, "A format compliant configurable encryption framework for access control of video," to appear in *IEEE Trans. CSVT, Special Issue on Wireless Video*, 2002.

[7] L. Qiao and K. Nahrstedt, "Comparison of MPEG encryption algorithms," *Inter. Journal on Computer & Graphics, Special Issue on Data Security in Image Communication and Network*, 22(3), 1998.

[8] A. S. Tosun and W. Feng, "A light-weight mechanism for securing multi-layer video streams," *Proc. IEEE Inter. Conf. on Information Technology: Coding and Computing*, pp. 157-161, April 2001.

[9] *ISO/IEC/SC29/WG11, "Information technology – Coding of audio-visual objects – Part 2: Visual ISO/IEC 14496-2"*, International Standards Organization, 11/98.

[10] P. Rogaway and D. Coppersmith, "A software-optimized encryption algorithm," *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 809, Springer-Verlag, pp. 56-63, 1994.

[11] MPEG4 IPMP FPDAM, ISO/IEC 14496-1:2001/ AMD3, ISO/IEC JTC 1/SC 29/WG11 N4701, March 2002

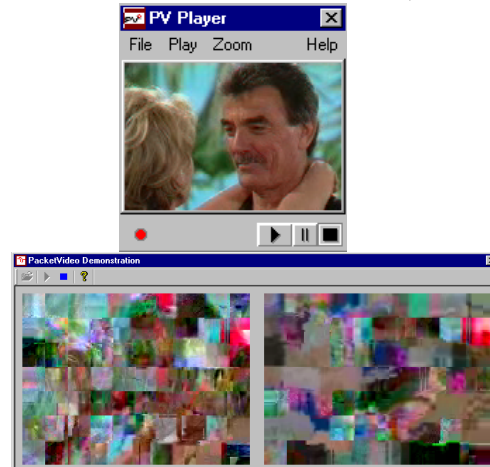


Fig. 1: One scrambled frame of "soap" sequence (5fps with const QP of 4), shuffling is based on video packet as a clear-text unit. Top: original unscrambled frame; BottomLeft: only MBs and 8x8 blocks are shuffled (using different shuffling tables); BottomRight: MBs, 8x8 blocks, and the first 10 run-level codewords in each block are shuffled separately.



Fig. 2: Attacked frame of Fig. 1 by setting DC of luminance component to 128, DC of chrominance components to 0, and all MV to 0.