Communication-Friendly Encryption of Multimedia

Min Wu and Yinian Mao
Electrical and Computer Engineering Department
University of Maryland, College Park, U.S.A.
Email: {minwu, ymao}@eng.umd.edu

Abstract— **This paper discusses encryption operations that selectively encrypt content-carrying segments of multimedia data stream. We propose and analyze three techniques that work in different domains, namely, a syntax-aware selective bitstream encryption tool with bit stuffing, a generalized index mapping encryption tool with controlled overhead, and an intra-bitplane encryption tool compatible with fine granularity scalable coding. The designs of these proposed encryption operations take into consideration the inherent structure and syntax of multimedia sources and have improved friendliness to communications, compression, and computation.**

I. Introduction

In the past decade, there has been significant progress in the coding and communications technologies for digital multimedia. The security of multimedia information, however, remains rather limited, which seriously curtails wider availability of multimedia information and the commercialization of information technologies. This paper focuses on protecting the confidentiality and achieving access control of multimedia information, which is one of the crucial security elements for many applications. Content confidentiality and access control is generally addressed by encryption, through which only authorized parties holding decryption keys can access content in clear text.

Traditional encryptions, which are primarily text based, treat data as a character string or a bit stream and generally do not utilize the inherent structures or the syntax of the data [1]. Perceptual multimedia data, such as image, video, and audio, distinguish themselves from generic data in that they can be represented in a lossy way with graceful quality degradation. The transmission of multimedia is no longer limited to delivering the exact version of a data stream from sender to receiver, but possibly involves such intermediate processing as bandwidth adaptation, unequal error protection, and transcoding in unicast, multicast, and broadcast scenarios over wired and wireless networks [2]-[4].

As illustrated in Fig. 1, there are two straightforward places to apply generic encryption to multimedia. The first possibility is to encrypt multimedia samples before any lossy or lossless compression (i.e. Stage#1 in Fig. 1). The main problem with this approach is that encryption often changes significantly the statistical and structural characteristics of the original multimedia source, resulting in much reduced compressibility. The second possibility is to apply encryption to the encoded bitstream after lossy or lossless compression (i.e., Stage#5 and #6 in Fig. 1) [5]. This approach introduces little overhead, but may scramble the structures and syntax readily available in the unencrypted bitstream. These structures, often indicated by special header/marker patterns or the context, would have enabled many processing desired in the intermediate network links, such as bandwidth adaptation via transcoding, unequal error protection, and random access [2]-[4][6]. The obvious approach of decrypting the stream, applying those processings, and then reencrypting the stream requires the encryption/decryption keys to be possessed by the possibly non-trustworthy intermediate links, which raises security concerns and is not desirable in many applications. Applying generic data encryption directly to multimedia information that often has large data volume also requires a tremendous amount of computation resources that is not be available in mobile devices [5].

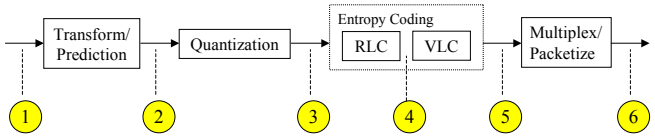

Fig. 1. Candidate domains to apply encryption to multimedia

Selective encryption, by which only a portion of the original data are scrambled, has been proposed to alleviate the problem of high computational complexity. By its name, selective encryption schemes only encrypt portions of multimedia data stream that carry rich content. For example, intra-coded frames and blocks of an MPEG video can be selected and encrypted using a classic DES cipher [7] or its variations [5], DC and selected AC coefficients in each block of a JPEG image or an MPEG video frame can be shuffled within the block[8] or across blocks [9], the codeword indices of the motion vectors can be encrypted by DES [6], and the Huffman codewords of coefficients and/or motion vectors can be encrypted by alternating among several Huffman codebooks in a cryptographically secure fashion [10]. Unfortunately, very few existing approaches are communication-friendly. And for those designed with communication issues in mind, there have been little analytic study on the tradeoffs such as the overhead in compressibility. Neither is there much comparison regarding encryption at different stages of multimedia representation process.

In this paper, we investigate possible domains at which encryptions can be applied. We consider quantized multimedia transform domain, intermediate bit planes, and bitstream domain. We propose and analyze three encryption operations, namely, a syntax-aware selective bitstream encryption tool with bit stuffing (Sec. II), a generalized index mapping encryption tool with controlled overhead (Sec. III), and an intra-bitplane encryption tool compatible with fine granularity scalable coding (Sec. IV). These operations can be used as building blocks and combined to form an encryption system for multimedia data. Our designs of these proposed encryption operations take into consideration the inherent structure and the underlying syntax of multimedia sources to achieve improved friendliness to communications, compression, and computation.

This research is supported in part by research grants from U.S. National Science Foundation CCR-0133704 (CAREER), Defense Advanced Research Projects Agency (DARPA) & Air Force Research Laboratory under Grant #F30602-00-0-0510, Army Research Office Grant #DAAD19-01-1-0494, and Minta Martin Foundation.

II. Syntax-Aware Selective Bitstream Encryption With Bit Stuffing

In this section, we discuss realizing selective encryption in bitstream domain (Stage #5 and #6 of Fig. 1). We have mentioned that headers and markers are special bit patterns in a compressed bitstream for indicating different logical units of the bitstream, providing side information, enabling random access, and assisting synchronization in error-prone transmission [4]. A natural way to realize selective encryption is to encrypt only the content-carrying fields of the multimedia bitstream, such as the fields of motion vectors and DCT coefficients in MPEG video, and keep the structure and headers/markers of the bitstream unchanged. Some parts of the encrypted multimedia data could, however, become identical to certain headers/markers, causing a potentially serious emulation problem as raised in [6] when the multimedia data go through certain network protocols, transcoding, and/or during error recovery. For example, the intermediate processing nodes may wrongly identify an encrypted version of a content-carrying segment as a header/marker thus retrieving, skipping, or dropping wrong fields. The content-carrying fields are often tightly interleaved with headers/markers whose occurrence in a bitstream is quite frequent. The introduction of an additional pair of markers indicating the beginning and the end of each encrypted segment will bring significant overhead in bitrate.

We propose to apply bit stuffing in conjunction with the above selective bitstream encryption to overcome the emulation problem. Let $[p_1, ..., p_n]$ be a bit pattern with which the encrypted content-carrying field is to avoid emulating. Whenever we encounter a segment in an encrypted content-carrying field $[b_{i+1}, b_{i+2}, ..., b_{i+n-1}]$ that is identical to $[p_1, ..., p_{n-1}]$, we insert a complement bit ${p_n}^c$ between b_{i+n-1} and b_{i+n}. This will guarantee no emulation of $[p_1, ..., p_n]$ in the encrypted content-carrying field. The end-user will perform inverse stuffing by deleting ${p_n}^c$ that immediately follows a pattern of $[p_1, ..., p_{n-1}]$, then perform decryption on the content-carrying fields. The selection of the pattern $[p_1, ..., p_n]$ should also avoid generating new emulation at the following bitstream segments after bit stuffing.

Assuming each encrypted bit is equally likely to be 1 or 0 and independent to each other, we can show that the expected relative overhead of using bit-stuffing to avoid emulating $[p_1, ..., p_n]$ is approximately $2^{-(n-1)}$, which is 0.8% for $n = 8$ and 3×10^{-3}% for $n = 16$. This implies that the overhead is almost negligible for reasonably large n. If there are more than one header or marker to be avoided emulation and if they do not share a relatively long prefix, we will need to perform bit stuffing with respect to each pattern and the expected overhead will increase linearly with respect to the number of such headers/markers.

The above calculation shows that bit stuffing technique prevents header/marker being emulated by encrypted content-carrying field at a cost of very small bitrate overhead. In such a selectively encrypted stream with bit-stuffing, if headers and markers alone are sufficient to indicate layered coding structures, error protection demands and/or content structures, intermediate processing units will then be able to parse these headers/markers from the selectively encrypted bitstream and use them to accomplish such tasks as bandwidth adaption (by ripping of enhancement layers), unequal error protection, and random access. It is, however, important to notice that while bit stuffing itself introduces little bitrate overhead, the intensive use of headers and markers bring a non-trivial amount of overhead to video encoding [4]. This implicit overhead should not be ignored in a fair comparison with other encryption approaches.

Additionally, format compliance is not preserved by syntax-aware selective bitstream encryption with bit stuffing. Taking MPEG video as an example, bit-stream domain encryption of motion vector fields and DCT coefficient fields will in general no longer be a valid sequence of valid codewords conforming to the MPEG standard. Bit stuffing will also lead to non-compliance in syntax. There have been interests recently on studying how to encrypt multimedia data in such a way that the encrypted data can still be represented in a meaningful, standard-compliant format [11]. Such encryption techniques are likely to emerge from elegant combinations and interplays between multimedia signal processing and contemporary cryptography. They are useful for secure multimedia communications that prefer handling standard-compliant streams and/or performing more sophisticated processing than what a reasonable number of headers/markers can help [6]. In the following sections, we propose an index mapping encryption tool generalized from a work by Wen et al. [6] and a new, bit-plane encryption tool compatible with fine granularity scalable coding. We are particularly interested in how much price in terms of compressibility these techniques have to pay to achieve standard-compliant encryption. We will answer this question through analysis and simulation.

III. Generalized Index Mapping with Controlled Overhead

A. Encryption via Index Mapping & Generalization

In [6], Wen et al. proposed a standard-compliant encryption operation by assigning to each variable-length-codeword (VLC) a fixed-length index, encrypting the indices using a standard block or stream cipher, and finally mapping the encrypted indices back to codeword domain. Wen's approach would work well with such codes as Huffman codes and Golomb-Rice codes, which associate each symbol in a finite set with a unique codeword of integer length. However, the application of this encryption tool to VLCs that allow fractional codeword length per symbol, such as the arithmetic codes, is difficult. In addition, analytic study is not available regarding the overhead in compressibility introduced by encryption.

The first problem can be overcome by extending the index encryption idea so that the encryption can be applied directly to symbols that take values from a finite set before getting into VLC codeword domain. This may include working with quantized coefficients, quantized prediction residues (Stage #3 in Fig. 1) and run-length coding symbols (Stage #4 in Fig. 1). As a simple example, we consider the symbols come from a finite set {A, B, C, D} and the symbol sequence to be encrypted is "ABBDC". We first assign a fixed-length index to each symbol: A→[00], B →[01], C→[10], D→[11]. We then convert symbol sequence to index sequence "00 01 01 11 10", and encrypt the index sequence using an appropriate core encryption algorithm such as a stream cipher with a one-time pad [0100 1011 1001 ...]. Finally we convert the encrypted index sequence "01 01 11 00 00" back to symbol sequence "BBDAA". After encryption, any appropriate VLC coding can be applied on the encrypted symbol sequence. Index encryption may also be extended from scalar to vector, where N symbols can be treated as a whole in index assignment and encryption, as long as the possible N-dimension vector values form a finite set. Block ciphers with

high cryptographic strength, such as the DES, the AES, or the RSA public-key encryption [1], can also be used as the core encryption algorithm to encrypt concatenated indices.

In the next subsection, we will address the second problem of the analysis and control of overhead.

B. Analysis & Control of Overhead

The encryption's impact on compressibility can be quantified by the changes in average code length before and after encrypting a sequence of symbols. Many multimedia standards provide default entropy codebooks, which are obtained from a set of representative signals and are used most of the time for the simplicity of implementations. We assume that the probability mass function of the symbol prior to encryption is $\{p_i\}$, that of the symbols after encryption is $\{q_i\}$, and the code length designed for distribution $\{p_i\}$ is $\{l_i\}$. If $\{q_i\}$ is a piecewise uniform approximation of $\{p_i\}$, i.e., for each subset S_j of a non-overlapped partition of symbol's range, we have $\sum_{i\in S_j} p_i = \sum_{i\in S_j} q_i = |S_j| q^{(S_j)}$ and $q_i = q^{(S_j)}$ for all $i \in S_j$, where $|\cdot|$ denotes the cardinality of a set. Assuming encryption changes the symbol distribution from $\{p_i\}$ to the above $\{q_i\}$ and the same codebook of code length $\{l_i\}$ is used both before and after encryption, we can show that the changes of the expected code length δL is

$$\delta L = \sum_i (q_i - p_i) l_i = D(p||q) + D(q||r) - D(p||r), \qquad (1)$$

where $D(\cdot||\cdot)$ represents the Kullback-Leibler divergence, and r denotes a probability distribution of $P(R=i) = 2^{-l_i} / \sum_k 2^{-l_k}$.

The rationale of the above formulation lies on the fact that encryption tends to flatten the probability distribution of a source: we can show that the ciphertext symbols from a truly random one-time pad is uniformly distributed regardless of the source's distribution before encryption; and uniform distribution is often a good assumption for ciphertext produced by many other contemporary ciphers. If encryption is done on an index representing the full range of symbol values, the distribution of ciphertext symbols, q, will be uniform over the entire range. Alternatively, as illustrated in Fig. 2, we can partition the range of symbol value into mutually exclusive subsets $\{S_j\}$ and restrict the encryption of a symbol $x \in S_j$ to be done within the subset, i.e., $Encrypt(x) \in S_j$. The distribution q will be a piecewise uniform approximation of p. Restricting the encryption output to be in the same subset as the input symbol, the complexity for brute force attack per symbol is reduced from $2^{C(S)}$ to $2^{C(S_j)}$. On the other hand, the overhead is also reduced because the distance from the piecewise uniform distribution q to the original distribution p is closer than that to a completely uniform distribution in the Kullback-Leibler divergence sense. Thus by controlling the set partitioning, we can have a tradeoff between the security and the overhead. In addition, Eq. 1 also indicates that the optimality of the codelength $\{l_i\}$ designed for probability distribution $\{p_i\}$ affects the changes of the expected code length after encryption. If the code is optimal for $\{p_i\}$, i.e., $l_i = -\log p_i$, then $D(p||r) = 0$ and Eq. 1 becomes $\delta L = D(p||q) + D(q||p)$.

We now use the DC coefficients of a JPEG-like representation of the 512×512 Lenna image to demonstrate the proposed approach. The DC coefficient in each block captures collectively the coarse information of an image and is differentially encoded to reduce the redundancy among them. For natural images, DC differential residues are approximately Laplacianly distributed with very small probability outside the range

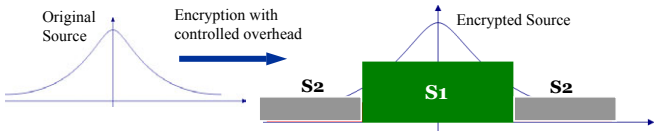

Fig. 2. Encryption via index mapping within subsets gives piecewise constant approximation of source distribution and controlled overhead.

of $[-63, 64]$. Before encryption, the average code length for encoding DCs is 5.78 bits. We apply the proposed generalized index encryption to the DC differential residues within $[-63, 64]$ without further set partitioning. The index encryption is realized via one-time pad, resulting in an average code length of 8.60 bits, or an increase of 2.82 bits. In a second test, we partition the symbol range of $[-63, 64]$ into two subsets $[-31, 32]$ and $[-63, -32] \cup [33, 64]$, and restrict the input and output of index encryption to be in the same subset. Fig. 3 shows the encryption result of the Lenna image [1]. With set partitioning, the overhead in average code length is reduced from 2.82 bits to 1.53 bits. In both cases the experimental results coincide with the analytic results from Eq. 1 within ± 0.01 bits.

Fig. 3. Encryption results on the Lenna image based on generalized index mapping of DC differential residues: (left) original, (right) encrypted.

IV. Intra Bit-plane Shuffling

Fine granularity scalability (FGS) is desirable in multimedia communications to provide a near-continuous tradeoff between bitrate and quality. FGS is commonly achieved by bit-plane coding, as used in the Embedded Zero-tree Wavelet (EZW) coder and the recently adopted MPEG-4 FGS coder [12]. We shall use MPEG-4 FGS as an example and the extension to other FGS coder is quite straightforward. As surveyed in [12], FGS is a functionality provided in the MPEG-4 streaming video profile. A video is first encoded into two layers, namely, a base layer that provides a basic quality level at low bit rate and an enhancement layer that provides refinement. The enhancement layer is encoded bitplane by bitplane from most significant bitplanes to least significant ones to achieve fine granularity scalability. Each bitplane within an image block is represented by (R_i, EOP_i) symbols, where R_i is the run of zeros before the i^{th} "1", and EOP_i is a flag indicating whether the current "1" is the last "1" bit in the current bitplane. The run-EOP symbols

[1] Encrypting DC alone is not secure enough as an attacker can still get the edge information by setting the DCs to constant and observing the resulting image. We encrypt DC only in this experiment for the purpose of demonstrating the proposed approach as one potential operations. A complete encryption system should encrypt both DCs and other information.

are encoded using variable-length codes and interleaved with sign bits in an elegant way.

To encrypt FGS encoded enhancement layers, we can apply the index-based encryption discussed in the last section to each run-EOP symbol. The overhead is rather independent from symbol to symbol and can be analyzed using Eq. 1. In the remaining part of this section, we propose an alternative encryption by shuffling each bit-plane according to a set of cryptographically secure shuffle tables. We will demonstrate that by allowing the joint consideration of symbols in the same bitplane, the encryption via intra-bitplane shuffling introduces smaller expected overhead.

More specifically, we perform random shuffling on each bitplane of n bits and the shuffled bitplane will then be encoded using run-EOP approach. For example, a bitplane "0 1 0 0 0 1 0 0 0 0" having $n_1 = 2$ bits of value "1" out of a total of $n = 10$ bits will lead to $\binom{n}{n_1} = 45$ different permutated patterns. An important property of shuffling is that the set of elements before and after shuffling are identical. For intra-bitplane shuffling, this implies that the number of ones is preserved hence the number of run-EOP symbols for representing the encrypted bitplane is unchanged, ensuring no overhead coming from the increase of run-EOP symbols. In addition, assuming for a specific n-bit bitplane, each of the $n!$ shuffles is equally likely, we can show that the expected number of occurrences that the run of zeros before a "one" equals to d is

$$E(N_d|n_1) = \left(\prod_{k=0}^{d-1}(1 - \frac{n_1}{n-k})\right)\frac{{n_1}^2}{n-d}. \qquad (2)$$

We therefore arrive at an expected histogram $\{E(N_d|n_1)\}$ indicating the likelihood of getting different zero-run length d. Shuffling can also be done in a larger spatial range than the run-EOP encoding, for example, we can shuffle the same bitplane of $n = 256$ bits within a macroblock and then perform run-EOP encoding in a smaller block of $n_B = 64$ bits. In this case, the expected zero-run histogram within an encoding block will become

$$E(N_d|n_1) = \prod_{k=0}^{d-1}(1 - \frac{n_1}{n-k}) \cdot \frac{n_1}{n-d} \cdot \left[n_1 - \frac{(n-n_B)(n_1-1)}{n-d-1}\right].$$

From the histogram before and after encryption we can obtain the expected overhead per symbol.

As a proof-of-concept, we experiment on the FGS encoded enhancement layer of two QCIF video sequences. One is the "Foreman" sequence with 10 frames, and the other is the "Carphone" sequence with 100 frames. We use intra bitplane shuffling to encrypt each bitplane of the enhancement bitstream, where shuffling is performed on a macroblock ($n = 256$) followed by encoding of blocks ($n_B = 64$). The expected histograms $\{E(N_d|n_1)\}$, presented in Fig. 4, show that the experimental results match the above analysis very well. The overhead for each bitplane is determined by the overhead of each symbol and the number of symbols in that bitplane (n_1). For the "Foreman" sequence, the relative overhead for encrypting the first three most significant bitplanes, which provides sufficient visual scrambling, is 7.0%, while the overhead by the index-mapping approach in Sec. III is 14.3%. The overhead reduction is more significant for lower bitplanes and when compared with applying generic encryption directly to bitplanes.

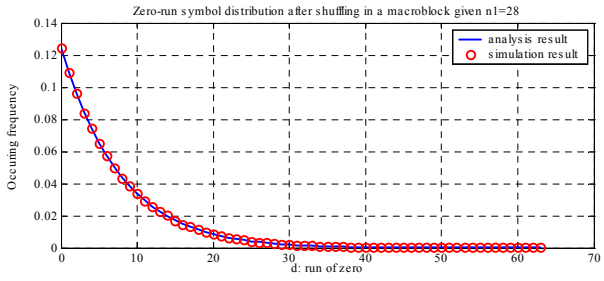

Fig. 4. Expected histograms of zero-run lengths after intra-bitplane shuffling: solid line indicates the analytic result, and circles indicate the experimental result; the number of "1"s per macroblock (n_1) is 28 bits, which is the average of the 2nd MSB bitplane of the "Foreman".

V. Conclusions

In summary, we studied selective encryption of multimedia with a focus on communication and compression issues. We identified a set of domains along the representation and communication process of multimedia where encryption can be applied, and proposed three encryption operations. We first showed that using bit stuffing technique, a bitstream domain selective encryption prevents header/marker emulation problem at a cost of little bitrate overhead. On the other hand, by moving the encryption domain from bitstream to upper levels and therefore preserving standard compliance, more sophisticated intermediate processing can be applied directly on the encrypted data. Under such a framework, we proposed an encryption tool via generalized index mapping, which can be applied to any scalar or vector symbols with a finite value range. The compression overhead can be adjusted and confined to a moderate amount. We also proposed and analyzed intra-bitplane shuffling, which is a promising encryption tool compatible with fine granularity scalable coding.

References

[1] W. Trappe and L.C. Washington: *Introduction to Cryptography with Coding Theory*, Prentice Hall, 2001.

[2] N. Yeadon, F. Garcia, D. Hutchison, D. Shepherd: "Continuous Media Filters for Heterogeneous Internetworking," *Proceedings of SPIE*, Multimedia Computing and Networking (MMCN'96), San Jose, Jan. 1996.

[3] M. Wu, R. Joyce, H-S. Wong, L. Guan, S-Y. Kung: "Dynamic Resource Allocation Via Video Content and Short-term Traffic Statistics", *IEEE Trans. on Multimedia*, vol.3, no.2, pp.186-199, June 2001.

[4] Y. Wang, S. Wenger, J. Wen, A. Katasggelos: "Error Resilient Video Coding Techniques", *IEEE Signal Processing Magazine*, vol.14, no.4, pp61-82, July, 2000.

[5] L. Qiao, K. Nahrstedt: "Comparison of MPEG Encryption Algorithms", *Inter. Journal on Computers & Graphics*, Permagon Publisher, vol. 22, no. 3, 1998.

[6] J. Wen, M. Muttrell, M. Severa: "Access Control of Standard Video Bitstreams", *Proc. of Inter. Conf. on Media Future*, Florence, Italy, May 2001.

[7] T-L. Wu, S.F. Wu: "Selective Encryption and Watermarking of MPEG Video", Inter. Conf. on Image Science, Systems, and Technology (CISST'97), Las Vegas, NV, 1997.

[8] L. Tang: "Methods for Encrypting and Decrypting MPEG Video Data Efficiently", *Proceedings of 4th ACM Inter. Conf. on Multimedia*, pp.219-229, Boston, MA, Nov. 1996.

[9] W. Zeng, S. Lei: "Efficient Frequency Domain Video Scrambling for Content Access Control", *Proc. of ACM Multimedia*, Orlando, FL, Nov. 1999.

[10] C-P. Wu, C.-C. Kuo: "Efficient Multimedia Encryption via Entropy Codec Design," in *SPIE Inter. Symposium on Electronic Imaging*, Proc. of SPIE, vol. 4314, San Jose, CA, Jan. 2001.

[11] MPEG-21 Part-4: Intellectual Property Management and Protection (working document), 2001.

[12] W. Li: "Overview of Fine Granularity Scalability in MPEG-4 Video Standard", IEEE Trans. on Circuits & Systems for Video Technology, vol.11, no.3, pp301-317, March 2001.