Hindawi Publishing Corporation
EURASIP Journal on Information Security
Volume 2007, Article ID 56365, 13 pages
doi:10.1155/2007/56365

Research Article

Efficient and Syntax-Compliant JPEG 2000 Encryption Preserving Original Fine Granularity of Scalability

Yang Yang,[1] Bin B. Zhu,[2] Shipeng Li,[2] and Nenghai Yu[1]

[1] Department of Electrical Engineering and Information Science, University of Science and Technology of China, Hefei, Anhui 230027, China
[2] Microsoft Research Asia, Beijing 100080, China

Correspondence should be addressed to Bin B. Zhu, binzhu@microsoft.com

Received 29 June 2007; Revised 29 October 2007; Accepted 23 November 2007

Recommended by E. Magli

A novel syntax-compliant encryption primitive and an efficient syntax-compliant JPEG 2000 encryption scheme are presented in this paper. The syntax-compliant encryption primitive takes, as input, syntax-compliant plaintext and produces syntax-compliant ciphertext. It is faster than all the other syntax-compliant encryption primitives we know. Our JPEG 2000 encryption scheme encrypts independently either each codeblock segment (normal mode) or each intersection of a codeblock segment and a codeblock contribution to a packet (in situ mode). Truncation-invariant parameters uniquely identifying each independently encrypted data block are combined with a global initialization vector to generate on the fly an initialization vector (IV) used to encrypt the data block. These IVs can be correctly regenerated even when the encrypted codestream is truncated. Encrypted codestreams are syntax-compliant. The original granularity of scalability is fully preserved after encryption so that an encrypted codestream can be truncated to adapt to different representations without decryption. Our JPEG 2000 encryption scheme is fast, error-resilient, and has negligible file-size overhead.

Copyright © 2007 Yang Yang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

JPEG 2000 [1, 2] is the state-of-the-art international standard for still-image compression. A key improvement over its predecessor JPEG is that JPEG 2000 provides highly flexible fine granularity scalability, enabling progressive transmission or refinement by quality, resolution, component, or spatial locality. This scalability makes it possible to *directly* truncate a single JPEG 2000 codestream into a large variation of "downscaled" yet valid and near rate-distortion-(RD-) optimal JPEG 2000 codestreams. As a comparison, to get a downscaled representation of a JPEG-compressed image, transcoding is needed: the codestream is first decoded, processed, and then re-encoded. The resulting codestream is no longer RD-optimal in general. This "encode once decode many ways" feature is very useful in many applications. For example, a single JPEG 2000 codestream can be stored on a downloading server for users with different display devices and network bandwidths to download from. A user with PC and high-speed bandwidth can download the original, high-resolution codestream. Another user with a mobile phone and slow network can download a version of properly reduced resolution.

In response to the growing need for protection of copyrighted images, Part 8 of the JPEG 2000 standard, commonly known as the secure JPEG 2000 or JPSEC [3], defines a standardized framework that different image protection tools can be applied to provide a number of security services such as confidentiality, integrity verification, source authentication. An overview of JPSEC can be found in [4].

Confidentiality is achieved through encryption. Besides typical security considerations, encryption of scalable multimedia such as JPEG 2000 compressed images has some additional requirements [5, 6]. One requirement is that an encrypted codestream should preserve the scalability of the unencrypted codestream as much as possible. With this feature, an encrypted codestream can be truncated directly by any party, possibly untrusted, without decryption to fit into a specific application. Another requirement is that an encrypted codestream should still be compliant to the syntax of the underlying format. Such an encrypted codestream can be correctly processed and decoded by encryption-unaware tools. For JPEG 2000, it means that a JPSEC encrypted codestream can be correctly decoded by an image viewer conformed only to Part 1, the core coding system, although the resulting display is likely garbled. Without syntax compliance, an old image viewer may not be able to process a JPSEC encrypted codestream or may even get crashed.

Early encryption schemes did not take into consideration either scalability or syntax compliance. Dang and Chau [7] proposed to apply the Data Encryption Standard (DES) [8] to encrypt the payload of each packet after packaging the codestream generated by the embedded zero-tree wavelet compression with the asynchronous transfer mode. The result is not syntax-compliant. Scalability is also lost after encryption. Later encryption schemes tried to preserve certain levels of scalability after encryption. Grosbois et al. [9] proposed two encryption schemes for JPEG 2000 to allow accesses to resolutions or layers without decryption, but the two access types cannot be supported with a single encrypted codestream. Another drawback is that the seed used to encrypt a codeblock is inserted after the codeblock's termination marker in order to be syntax-compliant. The seed may be lost during transmission or scheme-unaware truncation, resulting in an entire undecryptable codeblock. Wee and Apostolopoulos [10] proposed a secure scalable streaming (SSS) scheme which groups JPEG 2000 packets into SSS packets, and the payload of each SSS packet is independently encrypted with a block cipher in the cipher-block chaining (CBC) mode. The initialization vector (IV) used in encrypting the payload of a packet is inserted into the plaintext packet header. A major disadvantage of this scheme is that the scalability granularity is raised to the progressive SSS packet level. To reduce the encryption overhead introduced by inserted IVs in SSS packet headers, an SSS packet typically contains quite a few JPEG 2000 packets, resulting in a very coarse granularity of scalability after encryption.

Recent encryption schemes for JPEG 2000 ensure syntax compliance after encryption in addition to preserving scalability. This means an encrypted codestream is still a valid JPEG 2000 codestream that an encryption-unaware standard-compliant decoder can still decode, although the rendered result may be unintelligible. For JPEG 2000, syntax-compliant encryption requires that the ciphertext do not emulate any JPEG 2000 markers. In JPEG 2000, a bitstream produced by arithmetic coding does not contain any byte-aligned value between 0xFF90 and 0xFFFF, and a code-block contribution to a packet (CCP) cannot end with a byte 0xFF. Syntax-compliant encryption must ensure that the ciphertext does not contain any of those prohibited patterns. Wu and Ma [11] proposed a packet-level syntax-compliant encryption scheme which generates a pseudorandom sequence with bytes of value 0xFF discarded, and then encrypts each byte of the payload in a packet whose value is not 0xFF and whose preceding byte is not 0xFF by adding it with the corresponding byte from the pseudorandom sequence modulo 0xFF. As a result, the ciphertext will never contain a byte 0xFF, and thus is syntax-compliant. Wu and Deng [12] proposed to encrypt each code-block contribution to a packet (CCP) with a modular addition or a block cipher repeatedly until the ciphertext is syntax-compliant. Watanabe et al. [13] proposed another syntax-compliant encryption algorithm. The payload in each packet is divided equally into blocks. A single byte is selected from each block by a pseudorandom generator, and is checked for encryption. If the byte is below 0xF0, its lower half is selected; otherwise this byte is skipped. All selected half bytes are collected into a buffer, encrypted with a conventional cipher, and put back to the original locations. The resulting payload does not contain a byte 0xFF, and is therefore syntax-compliant. Recently, Fang and Sun [14] proposed a compliant encryption primitive to encrypt CCPs for JPEG 2000. In their primitive, plaintext and keystream are added bytewise modulo 0x100, 0x90, or 0x70, depending on the context and a running parameter. This compliant encryption primitive is designed for arithmetic coding pass bitstreams. It does not work directly for raw pass bitstreams where the arithmetic coder is bypassed (see Section 2 for details).

In [15], we have proposed a codeblock-based scheme which encrypts each codeblock independently. The scheme produces a small, about 1.0%, file-size overhead, and is not syntax-compliant. The scheme is further improved and made syntax-compliant. A short version of the improved scheme is published in a conference paper [16]. In this paper, we extend our conference paper [16] to a full paper with detailed description of a novel syntax-compliant encryption primitive and an efficient syntax-compliant encryption scheme for JPEG 2000 as well as more experimental results. Our syntax-compliant encryption primitive described briefly in [16] is called the Ciphertext Switching Encryption (CSE). CSE works for a general specification of syntax such as JPEG 2000 arithmetic coding pass bitstream syntax and raw pass bitstream syntax as well as MPEG-4 FGS syntax [17]. It generates syntax-compliant ciphertext of the same size as the plaintext, decryptable without any additional information except the decryption key. As we will see in Section 5, CSE is the fastest among all the existing syntax-compliant encryption primitives for JPEG 2000. The JPEG 2000 encryption scheme described in this paper is an improved version of that reported in the conference paper [16]. Our JPEG 2000 encryption scheme applies CSE to encrypt independently each codeword segment (normal mode) or each intersection of a codeword segment and a CCP (in situ mode). The scheme is syntax-compliant and preserves the original fine granularity of scalability of JPEG 2000. The scheme has two operation modes. In the first mode (i.e., the normal mode), each codeword segment is independently encrypted with CSE. In the second mode (i.e., the in site mode), the intersection of a codeword segment with a CCP is independently encrypted with CSE. An initialization vector (IV) is used for each independent encryption. A single global IV is inserted into the main header of JPEG 2000 codestream. The IV associated with an independent encryption is generated on the fly from the global IV and the unique index to the data block to be encrypted, resulting in a negligible file-size overhead. These indices are invariant when an encrypted codestream is truncated. As a result, a truncated codestream can still be correctly decrypted.

The rest of this paper is organized as follows. In Section 2, JPEG 2000 is briefly described, including selected details

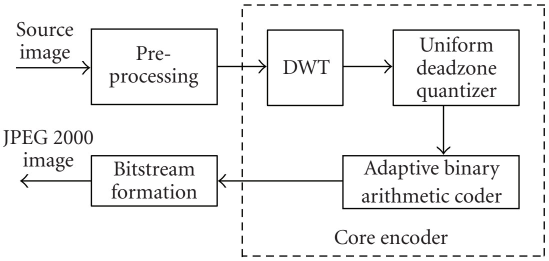

Figure 1: JPEG 2000 encoding process.

about JPEG 2000 and JPSEC. The ciphertext switching encryption primitive is described in Section 3, and our syntax-compliant JPEG 2000 encryption scheme is presented in Section 4. Experimental results are given and discussed in Section 5. We conclude the paper with Section 6.

2. INTRODUCTION TO JPEG 2000 AND JPSEC

JPEG 2000 is an image compression standard based on the discrete wavelet transform (DWT) and arithmetic coding. JPEG 2000 encoding can be roughly divided into 3 phases, preprocessing, encoding, and bitstream formation, as shown in Figure 1.

In the first phase, image pixels are preprocessed before sent into the core encoder. A source image is partitioned into nonoverlapping rectangular pixel blocks of arbitrary yet equal (except for those on the boundaries) sizes, known as tiles. A tile is the basic JPEG 2000 coding unit—each of them is encoded independently. Dividing image spatially into tiles reduces coding memory requirement and enables any part in the final codestream to be accessed and processed independently. An optional multicomponent transform is performed on each tile to decorrelate RGB color components into YC_bC_r or YUV components. Each color component is fed into the core encoder.

In the core encoder, an L-level DWT is applied to each color component of a tile to obtain $L + 1$ different resolution levels ranging from level 0 to level L, with the lowest frequency subband referred to as resolution level 0. Each resolution level contains three subbands except the resolution level 0 which contains only one subband. Resolution level 0 is referred to as resolution 0 of the tile-component. The tile-component at resolution $l > 0$ is the collection of the component at resolution $l - 1$ and the three subbands at resolution level l. The original tile-component is at resolution L. In the lossy encoding mode, all subband coefficients are then passed into a uniform dead-zone quantizer to reduce the data precision. Finally indices of the quantized coefficients of each subband are passed into the embedded block coding with optimized truncation (EBCOT) arithmetic entropy encoder, where each subband is divided into small rectangular blocks, called codeblocks, and each codeblock is independently encoded in a bitplane manner from the most significant to the least significant. Each bitplane is encoded in three sub-bitplane passes with the provision of truncating the bitstream at the end of each coding pass. The raw bits emitted from a coding pass are encoded with a context-adaptive binary arithmetic coder, namely, the binary MQ-coder, unless the lazy coding mode is used. In the lazy mode, the MQ-coder is entirely bypassed and the raw bits are emitted for certain coding passes. Details of the EBCOT arithmetic encoder can be found in [18].

A codeword produced by the above encoding process ends with codeword termination, which is essentially equivalent to stopping and restarting the encoder without necessarily resetting the states of its probability contexts. The standard requires that three specific termination patterns be supported [1, 2]. The default is to generate a single codeword segment for the entire codeblock, that is, terminating at the end of the last bitplane encoding of the codeblock. The second is to terminate at the end of every sub-bitplane coding pass so that the bitstream generated from each coding pass forms a codeword segment. The states of the probability contexts of the arithmetic encoder can be reset at the end of each coding pass to enable independent decoding of the bitstream of each coding pass. The last termination method is for the lazy coding mode that the boundary between the arithmetic coding passes and the raw passes must be terminated.

In the last phase, the outputs from encoding individual codeblocks are packed into a JPEG 2000 codestream whose structure is shown in Figure 2. A JPEG 2000 codestream starts with a main header, comprised of markers and marker segments, to provide the image parameters and coding parameters that can apply to every tile and tile-component. A marker segment is a marker and the associated set of parameters, used to indicate characteristics of the image. A tile-part header is inserted at the beginning of each tile-part codestream to provide the tile-part coding parameters. Six types of markers are defined in the JPEG 2000 standard Part 1 [1]. Each marker is two bytes long: the first byte is always 0xFF, and the second may have any value between 0x01 and 0xFE. An arithmetic codeword segment does not contain any byte-aligned value between 0xFF90 and 0xFFFF. A byte of 0xFF followed by any byte of value larger that 0x8F is recognized as a legitimate termination marker. A JPEG 2000 standard-compliant arithmetic decoder stops reading bytes when it encounters a termination marker, and all the data in a segment after the termination marker are therefore ignored. In a raw segment when arithmetic coding bypass is enabled, if a byte-aligned value is 0xFF, a single zero bit is stuffed into the most significant bit of the next byte. As a result, a byte of value 0xFF is never followed by a bit of 1 in a raw segment.

The bitstream of each codeblock is distributed across one or more layers in the codestream. Each layer consists of consecutive bitplane coding passes from each codeblock in a tile. A decoder is able to decode codeblock contributions contained in each layer successively so that the image quality improves progressively. JPEG 2000 also introduces a spatial-frequency construct called precinct, which is a collection of spatially contiguous codeblocks from all subbands at a particular resolution level. For a given tile, the data from a specific layer, a specific component, a specific resolution, and a specific precinct appears in the codestream as a continuous, byte-aligned segment referred to as a packet. The length of the data from each code block contribution to a packet

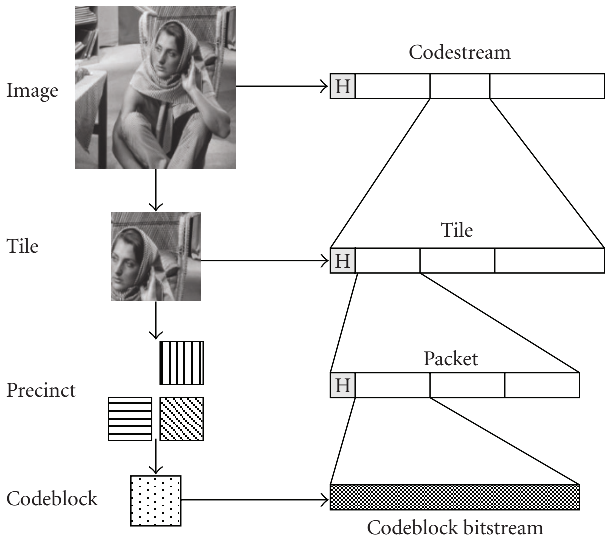

FIGURE 2: The structure of a JPEG 2000 image codestream.

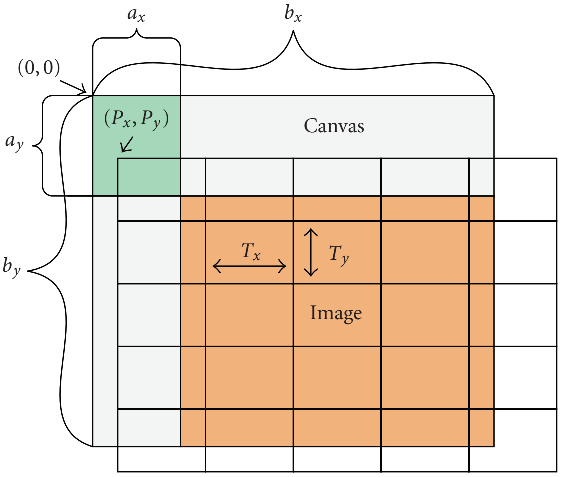

FIGURE 3: Canvas coordinates and tile partition in the high resolution grid.

(CCP) is indicated in the header of a packet. In the case of a CCP containing multiple codeword segments, the length of each codeword segment is indicated in the packet header. A CCP in JPEG 2000 never terminates in a byte of value 0xFF. The packets in a JPEG 2000 codestream can be arranged in different ways, called progression orders. The interleaving of the packets can progress along the four axes: layer, component, resolution, and precinct. Five progression orders are supported by JPEG 2000.

Errors may occur during transmission and storage. For example, random and burst bit errors may occur in wireless communications due to wireless characteristics, and packets may be lost in Internet communications due to traffic congestion. Arithmetic encoding in JPEG 2000 is a variable length coding which is susceptible to error propagation: a single bit of error in the compressed data may corrupt the decoding of the remaining bits, resulting in much more significant distortion in the decoded image. To confine error propagation and improve the quality of the decoded images under error-prone environments, JPEG 2000 provides error resilience bitstream syntax and tools [1]. The error resilience tools are defined on both the entropy encoding level and the packet level. On the former level, arithmetic encoding of the quantized coefficients is performed independently for each codeblock. As a result, bit errors are contained within the codeblocks where errors occur. In addition, termination of the arithmetic encoder and reset of the contexts are allowed after every coding pass. This enables the arithmetic decoder to continue decoding the coding passes after the coding pass containing errors. In the optional lazy mode, certain coding passes are not encoded by the arithmetic coder, and therefore are not susceptible to error propagation. A segmentation symbol can be encoded at the end of each bitplane to detect decoding errors. Correct decoding of this symbol confirms the correctness of the decoding of the corresponding bitplane. On the packet level, a resynchronization marker start-of-packet (SOP) together with a sequence number can be placed in front of every packet in a tile, allowing spatial partition and resynchronization of the codestream. Packets in a JPEG 2000 codestream can also be organized in the short packet format in which the packet headers are packed into either the main header or the tile-part header.

In JPEG 2000, an image is referenced from a canvas coordinate system [1, 2, 19]. As shown in Figure 3, the upper left-hand corner of the canvas is always located at the origin (0, 0) of the coordinate system. The right and lower boundaries of the canvas coincide with those of the image. An image is bounded by its upper left-hand corner of coordinates (a_x, a_y), and its bottom right-hand corner of coordinates (b_x, b_y). All tiles have exactly the same size $T_x \times T_y$ at the high resolution grid. Tiles on the boundaries may partially overlap with the image. Tiles are indexed incrementally from the first row to the last row and from the left to the right, starting from 0. Let (P_x, P_y) be the coordinates of the top left corner of the first tile on the high resolution grid. The JPEG 2000 standard mandates that

(1) $P_x, P_y \geq 0$;
(2) $P_x \leq a_x$ and $P_y \leq a_y$;
(3) $P_x + T_x > a_x$ and $P_y + T_y > a_y$.

Once the coordinates of the first tile is known, the coordinates of the remaining tiles can be derived. The coordinates of precincts are similarly defined. In addition, the coordinates of lower resolution grids can also be derived.

JPSEC introduces two new marker segments. One is SEC (of value 0xFF94) in the main header used to carry overall information about the security tools and parameters applied to the image. The other is INSEC (of value 0xFF95) in the bitstream to provide information of localized security tools and parameters.

Further details about JPEG 2000 and JPSEC can be found in [1–4, 19, 20].

3. CIPHERTEXT SWITCHING ENCRYPTION

In this section, we present our novel syntax-compliant encryption scheme, the ciphertext switching encryption (CSE), which is used to encrypt JPEG 2000 codestreams syntax-compliantly. Like the iterative encryption scheme proposed by Wu and Deng [12], CSE can be applied to ensure ciphertext compliant to a general specification of syntax, such as JPEG 2000's arithmetic coding pass bitstream syntax and raw pass bitstream syntax presented in this paper and MPEG-4 FGS syntax presented in [17]. On the contrary, the syntax-compliant encryption schemes described in [11, 13, 14] are applied only for a specific syntax, that is, the JPEG 2000's arithmetic coding pass bitstream syntax.

CSE works with a stream cipher. Postprocessing operations are added to a conventional stream cipher encryption to ensure syntax compliance. More specifically, illegal bitstrings in the intermediary ciphertext generated by the conventional stream cipher are replaced by the corresponding bitstrings from the plaintext to force the ciphertext syntax-compliant. A bitstring is defined in this paper as a minimum number of consecutive bits starting at the boundary of a byte (i.e., byte-aligned) that are checked for compliance. In JPEG 2000, an arithmetic codeword segment does not allow any two consecutive bytes of value between 0xFF90 and 0xFFFF. To check syntax compliance against this requirement, a bitstring is two consecutive bytes. A raw segment, on the other hand, does not allow a byte 0xFF followed by a bit 1. A bitstring in this case is a byte plus the following bit, 9 bits in total. JPEG 2000 does not allow ao CCP ending with a byte 0xFF. A bitstring in this case is therefore one byte. The effective result after postprocessing is that some ciphertext bits are "switched" back to the plaintext bits. The tricky part in switching operations is to ensure that the decryptor can locate the switched bitstrings and correctly decrypt the ciphertext without any additional information other than the ciphertext and the decryption key. Our experimental results to be reported in Section 5 will show that only a very small percentage of ciphertext bits are required to switch back to the plaintext bits.

In describing CSE, an uppercase letter without subscript indicates a string of bytes, and a lowercase letter with a subscript index indicates the byte specified by the index in the string denoted by the corresponding uppercase letter. An uppercase letter with a subscript index indicates a bitstring or the remaining bits if there are not enough bits to construct a bitstring, starting at the byte specified by the index. Note that the ending index of a bitstring is not indicated in this representation since different syntaxes may have different numbers of bits in a bitstring, and a bitstring may end with a partial byte, although it always starts at a byte boundary. For example, an internal bitstring in the JPEG 2000 raw segment syntax consists of one byte plus the following bit, that is, 9 bits in total. Let $M = m_0 \| m_1 \| \cdots \| m_{n-1}$ be a syntax-compliant plaintext of n bytes, where m_i denotes one plaintext byte and $\|$ means concatenation. Let $C = c_0 \| c_1 \| \cdots \| c_{n-1}$ be the corresponding syntax-compliant ciphertext of exactly the same length as the plaintext, where c_i denotes one ciphertext byte. CSE works as follows.

(1) Generate a keystream $S = s_0 \| s_1 \| \cdots \| s_{n-1}$ of n bytes.

(2) Let $R = r_0 \| r_1 \| \cdots \| r_{n-1}$. Calculate $R = M \oplus S$, that is, $r_i = m_i \oplus s_i$, $0 \leq i < n$, where "$\oplus$" is the bitwise XOR operation.

(3) Initialize the current byte index *currIdx* to 0: *currIdx* = 0, and index *lastModIdx* of the rightmost switched bitstring to -1: *lastModIdx* = -1.

(4) Run *currIdx* from the current byte to the last byte $n-1$ to search for illegal bitstrings in R. If an illegal bitstring is found at *currIdx*, go to step (5). Otherwise, go to step (9).

(5) Replace the illegal bitstring $R_{currIdx} = r_{currIdx} \| \cdots$ starting at *currIdx* with the corresponding plaintext bitstring $M_{currIdx} = m_{currIdx} \| \cdots$, $R_{currIdx} \leftarrow M_{currIdx}$. Set the index *justModIdx* of the first byte of the just switched bitstring $R_{justModIdx}$ to be *currIdx: justModIdx* = *currIdx*.

(6) Check the decrypted bitstrings backward as follows:

   (i) for *backIdx* = *justModIdx*$-1 : -1 :$ *lastModIdx*$+1$ with the bitstring $R_{backIdx}$ overlapping with the bitstring $R_{justModIdx}$;

   (ii) set $D_{backIdx} = R_{backIdx} \oplus S_{backIdx}$, and switch the bits in $D_{backIdx}$ back to those in M if those bits were switched previously and their locations do not overlap with any bits in $R_{justModIdx}$;

   (iii) if $D_{backIdx}$ is an illegal bitstring, replace all the bits from $R_{backIdx}$ to $R_{justModIdx}$ with the corresponding plaintext bits: $r_i \leftarrow m_i$, $backIdx \leq i < justModIdx$, set *justModIdx* = *backIdx*, and go back to step 6(i).

(7) Check the ciphertext backward as follows:

   (i) for *backIdx* = *justModIdx* $-1 : -1 :$ *lastModIdx*$+1$ with the bitstring $R_{backIdx}$ overlapping with the bitstring $R_{justModIdx}$;

   (ii) if $R_{backIdx}$ is an illegal bitstring, replace all the bits from $R_{backIdx}$ to $R_{justModIdx}$ with the corresponding plaintext bits: $r_i \leftarrow m_i$, $backIdx \leq i < justModIdx$, set *justModIdx* = *backIdx*, and go back to step 7(i).

(8) If no bitstring is replaced in both steps (6) and (7), set *lastModIdx* = *currIdx* and *currIdx* = *currIdx* + 1, and go back to step (4). Otherwise, go back to step (6).

(9) Output R as the syntax-compliant ciphertext $C : C = R$.

The CSE encryption operations are shown in Figure 4. Like conventional XOR-based string ciphers, CSE decryption is identical to CSE encryption. Therefore, the operations described above and shown in Figure 4 are applied in both CSE encryption and decryption. The first two steps in CSE encryption are operations of a conventional stream cipher encryption. The remaining steps are the postprocessing steps to switch ciphertext bits back to the corresponding plaintext bits when necessary to ensure syntax compliance. The tricky part is to ensure that process can be reversed to recover the plaintext without any additional information other than the ciphertext. Once an illegal bitstring is found in the intermediary ciphertext R and switched with the plaintext bitstring in step (5), backward checking is applied in both steps (6) and (7) to ensure that preceding ciphertext bitstrings are still syntax-compliant and the just switched bitstring can be located in decryption. In step (6), the preceding ciphertext bitstrings are decrypted and checked to make sure that the just switched bitstring can be correctly identified and switched, and no spurious switching would occur during the decryption process. This decryption is done by XORing the intermediary ciphertext with the keystream, and then switching the bits that have been switched during the previous encryption operations except those bits with locations overlapping with the just switched bitstring. Switching a bitstring may make preceding ciphertext bitstrings illegal. Step (7) checks preceding ciphertext bitstrings to ensure that they are syntax-compliant. Switching operations may be applied when needed. In both steps (6) and (7), only the bitstrings overlapping with the just switched bitstring need to be checked since other preceding bitstrings are not affected by the bitstring just switched. steps (6) and (7) are iteratively applied until there is no more switching in both steps. Step (8) checks this condition. If no switching occurs in both steps (6) and (7), it goes back to step (4) to check the remaining bytes.

To facilitate understanding of CSE, an example of CSE encryption is given here. Suppose that $n = 8$ and the plaintext is $M = m_0 \| m_1 \| \cdots \| m_7 = 0\text{x}\,00\,11\,\text{FF}\,66\,\text{FF}\,66\,99\,22$, which is compliant with JPEG 2000 syntax. Let the keystream be $S = s_0 \| s_1 \| \cdots \| s_7 = 0\text{x}\,33\,44\,66\,99\,99\,99\,00\,55$. The intermediary ciphertext after step (2) is $R = M \oplus S = 0\text{x}\,33\,55\,99\,\text{FF}\,66\,\text{FF}\,99\,77$. In step (4), the first illegal bitstring is found at *currIdx* = 5 : $R_5 = 0\text{x}\,\text{FF}\,99 \geq 0\text{x}\,\text{FF}\,90$. In step (5), the illegal bitstring R_5 is switched to the plaintext bitstring: $R_5 = M_5 = 0\text{x}\,66\,99$. Backward checking is then conducted in steps (6) and (7). In step 6(ii), a preceding bitstring D_4 is decrypted to be illegal: $D_4 = 0\text{x}\,\text{FF}\,\text{FF}$. Step 6(iii) switches R_4 back to the plaintext bitstring: $R_4 = M_4 = 0\text{x}\,\text{FF}\,66$. In the subsequent backward checking of ciphertext, the ciphertext bitstring $R_3 = 0\text{x}\,\text{FF}\,\text{FF}$ is no longer legal. Step 7(ii) switches the bitstring to the corresponding bitstring: $R_3 = M_3 = 0\text{x}\,66\,\text{FF}$. In step (8), since switching has occurred in both steps (6) and (7), CSE goes back to step (6) to check backward the decrypted bitstrings again starting from D_2. In step 6(ii), $D_2 = 0\text{x}\,99\,66 \oplus 0\text{x}\,66\,99 = 0\text{x}\,\text{FF}\,\text{FF}$ is calculated. No bits in the bitstring need to be switched. This decrypted bitstring $D_2 = 0\text{x}\,\text{FF}\,\text{FF}$ is illegal. In step 6(iii), R_2 is therefore switched to the plaintext bitstring: $R_2 = M_2 = 0\text{x}\,\text{FF}\,66$, resulting in the intermediary ciphertext as $R = 0\text{x}\,33\,55\,\text{FF}\,66\,\text{FF}\,66\,99\,77$. No more switching occurs when steps (6) and (7) are applied. In step (8), CSE sets *lastModIdx* = 5 and *currIdx* = 6, and goes back to step (4) to continue processing the remaining bytes. No more illegal bitstrings are found. The final ciphertext is therefore: $C = 0\text{x}\,33\,55\,\text{FF}\,66\,\text{FF}\,66\,99\,77$, which is syntax compliant. The same procedure can be applied to the ciphertext C to recover the plaintext. Interested readers can apply the procedure to C to confirm that the decrypted plaintext is indeed identical to original plaintext. More details of CSE can be found from a C++ implementation available online [21]. That implementation is for easy understanding. A faster C++ implementation of CSE for JPEG 2000 was used in our experiments reported in Section 5.

Theoretical analysis of CSE's security is complex and out of the scope of this paper. Only a discussion is given here. Like other syntax-compliant encryption schemes such as those reported in [11, 13, 14], CSE does not meet indistinguishability under a chosen-plaintext attack (IND-CPA) security [22]. For an IND-CPA security encryption scheme, an adversary cannot tell which of two chosen messages with the same length is encrypted, that is, the ciphertext does not leak any information about any chosen plaintext. For the schemes reported in [11, 13, 14] as well as CSE, different operations are applied to a group of data, depending on its context in plaintext and/or ciphertext domain, resulting in possibly different distributions which can be used to tell which chosen message is encrypted. Fortunately, IND-CPA security is not normally necessary in practical multimedia encryption. In CSE, the intermediary ciphertext after the first two steps, that is, encryption by a conventional stream cipher, is random if the stream cipher is secure. This implies that the locations of switched bitstrings are random and cannot be predicted even if the plaintext is known. Under known plaintext attacks, there is a possibility that an adversary may identify the switched bitstrings by comparing the plaintext and ciphertext and exploiting the fact that a switched bitstring has a higher probability than a random bitstring to match that in the plaintext, but the adversary would not be able to deduce the encryption key if the underlying stream cipher is secure. From the experimental results reported in Section 5, for plaintext of 5000 bytes, about 16 bytes, that is, 0.32% of the ciphertext is switched to the plaintext. Each sequence of switched bits is about 2 bytes long. Since encryption is applied to compressed bitstreams in our applications, and compression removes most redundancy in a multimedia signal, CSE leaks very little information about the encrypted content. Even if an adversary could successfully identify the switched bitstrings, he could only gain some information about the restrictions on the neighboring bitstrings overlapping the switched bitstrings, without being able to deduce the neighboring plaintext bytes or other nonswitched plaintext bytes, or the encryption key. In conclusion, CSE is sufficiently secure for the targeted applications.

4. SYNTAX-COMPLIANT JPEG 2000 ENCRYPTION SCHEME

4.1. *Overview of our scheme*

The main goal in the design of our JPEG 200 encryption scheme is to ensure that the encrypted codestream is syntax-compliant and preserves the original granularity of scalability of JPEG 2000. Encryption may lower the compression efficiency or add additional data or markers to a codestream for correct decryption. It may also deteriorate the perceptual quality of the rendered image more than that without encryption when error or loss of data occurs, due to additional error propagation attributed to encryption. An additional

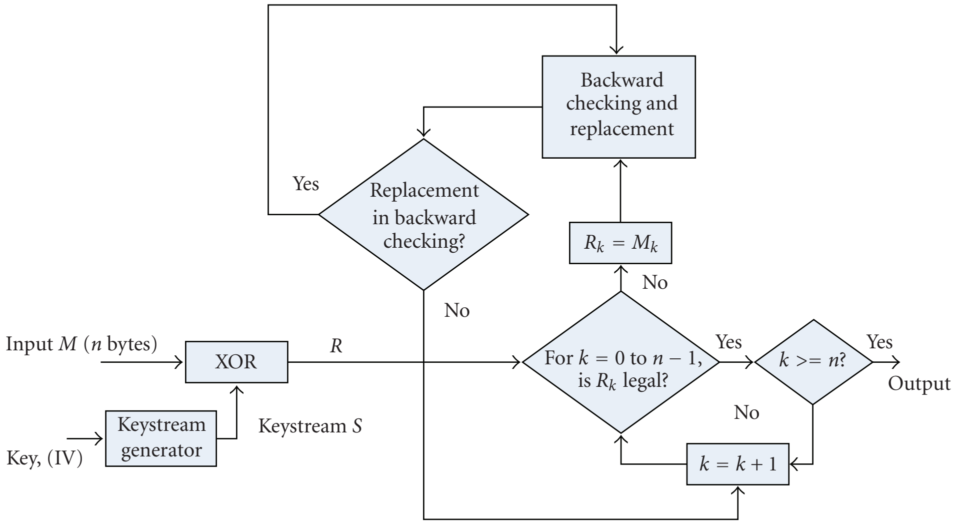

Figure 4: The operations of the ciphertext switching encryption primitive.

goal in our design is to minimize the perceptual degradation attributed to encryption's error propagation as well as the file-size overhead.

To ensure syntax-compliant, our scheme applies CSE to encrypt only the compressed data in each packet while leaves the main, tile-part, and packet headers untouched. Since the encryption process is applied after the compression process, the compression efficiency is not affected. To minimize the perceptual degradation attributed to encryption's error propagation, the data affected by the error propagation attributed to encryption should overlap as much as possible with the data affected by the error propagation attributed to the JPEG 2000 compression. In other words, the boundary of independent encryption should coincide with the termination of the arithmetic coding for error resilience. Recall that in JPEG 2000, each codeblock is independently encoded in a bitplane manner from the most significant to the least significant, and there are three termination patterns for encoding a codeblock. The arithmetic decoding terminates at the end of an arithmetic codeword segment. When a bit error occurs, arithmetic decoding of the remaining bits in the same codeword segment will be incorrect, but decoding of the compressed data in other codeword segments is not affected. (For the subsequent codeword segments of the same codeblock, the states of the probability contexts of the arithmetic encoder should be reset at the end of a codeword segment to enable independent decoding of the compressed data in these subsequent codeword segments.) As a result, encrypting each codeword segment independently would minimize the perceptual degradation attributed to encryption's error propagation. This is exactly what we have adopted in our scheme. As discussed in detail in Section 4.4, such encryption also preserves the finest granularity of scalability of a JPEG 2000 codestream, enabling the most flexible truncation of an encrypted JPEG 2000 codestream without decryption.

Each independent encryption requires some encryption parameters such as an initialization vector (IV) in our scheme. Insertion of IVs into an encrypted codestream would increase the file-size overhead. To reduce the adverse impact on the file-size, we need to generate those IVs on the fly so that we do not need to store them in an encrypted codestream. A major challenge to achieve this goal is that generation of IVs has to be invariant when a codestream is truncated in an arbitrary yet allowed manner. Otherwise wrong IVs may be generated when truncation of an encrypted codestream occurs. In our design, a set of truncation-invariant parameters uniquely identifying a codeword segment is used to generate the IV used to encrypt the codeword segment.

An alternative approach is to encrypt the intersection of a CCP and a codeword segment independently so that each CCP can be encrypted or decrypted in situ, a desirable feature when the encryption or decryption process interleaves with the encoding or decoding process. The two approaches are taken as two operation modes in our JPEG 2000 encryption scheme. They are referred to as the normal mode and the in situ mode, respectively, in this paper.

4.2. Generation of truncation-invariant IVs

A distinct IV is generated on the fly for each independent encryption. A global IV is inserted into the SEC maker segment in the main header of an encrypted JPEG 2000 codestream. This global IV is combined with the unique identifier of each independently encrypted data block to generate the IV to encrypt the data block. In the normal encryption mode, each codeword segment is independently encrypted. In the in situ mode, each intersection of a CCP and a codeword segment is independently encrypted. In both modes, we need to uniquely identify each codeblock. The codeblock identifier used to generate the IVs should be available from the JPEG 2000 codestream so that we do not need to store in the encrypted codestream, and also must be invariant under allowed truncations to a JPEG 2000 codestream so that the same IVs will be regenerated even when an encrypted

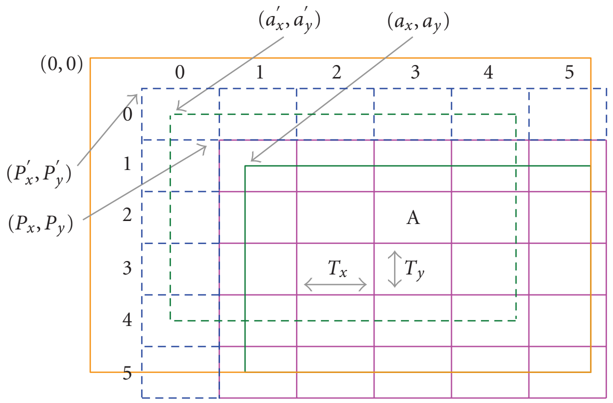

Figure 5: The extended JPEG 2000 canvas coordinates. Solid line rectangles represent the canvas, the image area, and the tiling grid. Dashed line rectangles represent the padded tiles.

codestream undergoes an arbitrary yet allowed truncation. To identify a codeblock, we need to identify the tile that the codeblock belongs to. Unfortunately, the tile index used in a JPEG 2000 codestream to identify a tile is not truncation-invariant. When some tiles are truncated from a JPEG 2000 codestream, such as truncating a JPEG 2000 codestream to crop the image from aspect ratio 16 : 9 to 4 : 3, the surviving tiles are reindexed starting from 0, resulting in indices inconsistent with the original ones. We exploit in our scheme the property that the canvas coordinates are invariant under allowed truncations to define a unique and truncation-invariant identifier for a tile.

To achieve this goal, the image and its tiles are extended upwards and leftwards by padding towards the origin a multiple number of tiles. More precisely, the extended image is first initialized to be the original image: $(a'_x, a'_y) = (a_x, a_y)$ and $(P'_x, P'_y) = (P_x, P_y)$. Then the following iterative moves are applied to extend the image: in each move, the coordinates (a'_x, a'_y) of top left corner of the extended image and the coordinates (P'_x, P'_y) of the top left corner of the extended first tile are reduced by subtracting T_x and the T_y in x- and y-directions, respectively, where $T_x \times T_y$ is the size of a tile at the high resolution grid, until the following conditions are satisfied:

(1) $P'_x, P'_y \geq 0$ and $a'_x, a'_y \geq 0$;
(2) $P'_x < T_x$, $P'_y < T_y$, and $P'_x + T'_x > a'_x$, $P'_y + T'_y > a'_y$;
(3) $P'_x \leq a'_x$, $P'_y \leq a'_y$.

Figure 5 shows an example of the actual image and its tiles (solid lines) as well as the extended image and the extended tiles (dotted lines).

After the above extension of the image and its tiles, each tile is indexed, with both the actual and the extended (or virtual) tiles counted, starting from 0 along both x- and y-directions, from top to bottom and from left to right. These indices are also shown in Figure 5. Such a tile coordinate system is used in our scheme to identify each tile uniquely for an image. For example, tile "A" shown in Figure 5 is uniquely identified by its coordinates (3, 2), that is, its index along the x-direction is 3, and along y-direction is 2. It is easy to check that the resulting indices of a tile in the proposed tile indexing system do not change even if an arbitrary combination of tiles are cropped from a JPEG 2000 codestream. Therefore, they are invariant for allowed truncations to a JPEG 2000 codestream.

A precinct can also be uniquely identified by the canvas coordinates of their top left-hand corners, which are truncation-invariant. With these truncation-invariant tile and precinct identifiers, a codeblock can be uniquely identified by the tile, component, resolution level, precinct, and the subband it belongs to, and the coordinates of codeblock's top left-hand corner on the canvas grid. Such a codeblock identifier is truncation-invariant.

The IV for each independent encryption is generated as follows. In the normal encryption mode, the global IV, the unique codeblock identifier defined above, and the index of the first coding-pass in a codeword segment are concatenated and hashed, truncated if necessary, to generate the IV used to encrypt the codeword segment. In the in situ mode, the global IV, the unique codeblock identifier defined above, and the index of the first coding-pass in an intersection of a codeword segment and a CCP are concatenated and hashed, truncated if necessary, to generate the IV used to encrypt the intersection. Apparently, the IVs generated in this way do not change when an encrypted codestream is truncated. In typical cases, the concatenation of a codeblock identifier and a coding pass index can be fit into the length of an IV. In this case, the concatenation is XORed with the global IV in generating the IV, and no hashing operation is needed.

In our scheme, a single global IV in the SEC header is the only information needed to be added to an encrypted JPEG 2000 codestream. Since encryption is applied after compression and the ciphertext has the same length as the plaintext with CSE, we can expect that the file-size overhead of our scheme is negligible. This is confirmed by our experiments reported in Section 5.

4.3. Encryption, partition, and CSE's file-size overhead

With the IVs generated by the method described in Section 4.2, each codeword segment or intersection of a codeword segment and a CCP is encrypted syntax-compliantly with CSE in the normal or the in situ mode. Note that the output has the same length as the input for CSE. In JPEG 2000, a codeword segment contains a number of complete coding passes, that is, a coding pass will not split into two codeword segments. By using the coding pass length calculation algorithm given in Annex D of the JPEG 2000 standard Part 1, no coding pass will ever be considered as terminating with an byte 0xFF [1]. A CCP does not end with a byte 0xFF either. As a result, no matter which encryption mode is used, the ciphertext will not end with a byte 0xFF in our scheme.

In JPEG 2000, each CCP contains either none or a number of coding passes of a codeblock. Recall that there are three termination patterns in JPEG 2000. When the arithmetic coding terminates at the end of each coding pass, there is no difference between the two encryption modes for our scheme since each coding pass forms a codeword segment. For the other two termination patterns, the two modes may operate differently. If a CCP contains multiple codeword segments, the two modes still operate in the same way: each codeword segment is encrypted independently. When a CCP contains a single codeword, such as when the default termination pattern is used, the two encryption modes operate differently. The in situ mode encrypts the CCP independently in this case, while an encrypted codeword segment may split into two or more CCPs in the normal mode encryption. Although a coding pass will not end with a byte 0xFF, a nonending coding pass in a codeword segment may end with 0xFF after CSE encryption. In addition, some ciphertext bits are switched with the corresponding plaintext bits in CSE. As a result, a partition method is needed to distribute the ciphertext from an encrypted codeword segment into different packets such that each CCP of the codeblock terminates at a right position so that decryption can be executed correctly even if the codeword is truncated at the boundary of a CCP. This requires that a sequence of switched bits could not be split into two CCPs. They have to stay in a single CCP. It also implies that CCP cannot end with a byte 0xFF for an encrypted bitstream either.

Since CSE is basically an XOR-based stream cipher with some ciphertext bits switched with the corresponding plaintext bits when needed, each byte in the ciphertext corresponds to the byte at the same position in the plaintext. Our partition method for an encrypted codeword segment is based on the original partition method used in JPEG 2000. When the original partition method is applied to the plaintext, if a corresponding partition position in the ciphertext does not end with a byte 0xFF or in the middle of a sequence of switched bits, then the partition position is also applied to the ciphertext, resulting in no file-size overhead for this partition position. Otherwise, the partition position is moved to the following byte or bytes such that the current CCP does not end with a byte 0xFF and a sequence of switched bits is not split into two CCPs, resulting in file-size overhead since when truncated at the boundary of the current CCP, the encrypted codestream contains more bytes (the additional bytes are dropped in decompression if truncated at the current CCP) than the case without encryption for the same rendered image.

A sequence of switched bits is typically two bytes long for CSE encryption of an arithmetic codeword, as indicated by the experiments in Section 5. This means that when adjustment to a partition position is needed, the CCP boundary is typically moved to the next byte, resulting in one byte overhead for the CCP. Since the chance that an encrypted CCP ends with a byte of value 0xFF or in the middle of switched bits is slim, the incurred file-size overhead for CSE is almost negligible. The CSE encryption in the normal mode does not incur any file-size overhead under other cases. As a comparison, the CSE encryption in the in situ mode does not incur any file-size overhead.

In the in situ encryption mode, each CCP can be read into a buffer, partitioned into codeword segments if it contains more than one codeword segment, and then CSE is applied to each codeword segment or the CCP. The encryption result, which has exactly the same length as the CCP, is then placed back to the original position. The process does not affect other data. Therefore, the encryption and decryption in this mode can be applied in situ to the CCP buffer. As a comparison, in the normal mode, the size of a CCP may be changed, for example, when the CCP of ciphertext ends with a byte 0xFF or in the middle of a sequence of switched bits, resulting in a modified packet header and packet size. The in situ property might be desirable in some applications.

4.4. Scalability and error resilience

Our JPEG 2000 encryption scheme produces an encrypted, syntax-compliant codestream, and preserves the original granularity of scalability. It is obvious that the original granularity of the five types of scalabilities in JPEG 2000, that is, tile, component, resolution, precinct, and layer, is fully preserved in an encrypted codestream with our scheme. An encrypted codestream can also be truncated to the smaller level of a codeword segment and a CCP, the same level as supported by JPEG 2000.

In addition to the scalability to allow truncations, a JPEG 2000 codestream can be repackaged by splitting or combining packets. A CCP may contain multiple codeword segments. A packet containing CCPs with multiple codeword segments can be split into multiple packets since the size for each codeword segment can be derived from the packet header. After encryption with our scheme, such a packet can still be split into multiple packages without decryption since our scheme encrypts independently each codeword segment or each intersection of a codeword segment and a CCP. When our scheme runs in the normal mode, combining two or more packets into one packet is still allowed after encryption since each codeword segment is independently encrypted. This is not true when our scheme runs in the in situ mode since the ciphertext in a CCP appended to another CCP after combining will not be correctly decrypted: a wrong IV is used in decryption. Therefore, the original repackaging capability is preserved when our JPEG 2000 encryption scheme runs in the normal mode but only package splitting is preserved when in the in situ mode.

The syntax-compliance and preservation of the original granularity of scalability allow an encryption-unaware processor to process, for example, to truncate an encrypted codestream in the same way as a nonencrypted codestream without decryption. There is no need to use any decryption secrets in the process. As a result, the processor does not need to be trusted.

As a comparison, Dang and Chau encryption scheme [7], Wu and Ma scheme [11], and Watanabe's scheme [13] raise the granularity of scalability to the packet level, and Wu and Deng scheme [12] and Fang and Sun scheme [14] to the CCP level, resulting in a granularity of scalability coarser than that offered by JPEG 2000 after encryption. Note that a CCP may contain multiple codeword segments. Our scheme has finer granularity of scalability than those JPEG 2000 encryption schemes.

When a bit error occurs in a codeword segment of a codeblock, the erroneous coding pass containing the error bit and all subsequent coding passes in the same codeword segment are rendered undecodable. In our scheme, each codeword segment or each intersection of a codeword segment and a CCP is encrypted independently, and a syntax-compliant stream cipher is used as the encryption primitive. If the bit error removes or generates spurious ciphertext switching incidences, our encryption does incur error propagation: the error expands to the whole sequence of switched bits, typically 2 bytes. Fortunately, switching occurs infrequently, as shown from the experimental results reported in Section 5. As a result, our encryption scheme has a very small if not negligible adverse impact on the perceptual quality when error or loss of data occurs, which is confirmed by our experiments also reported in Section 5. In other words, our encryption scheme is error-resilient. This has been confirmed by our experiments reported in Section 5.

As a comparison, a bit error may cause a wrong selection of bytes to be encrypted in Watanabe's scheme [13], resulting in wrong decryption of a whole packet. A bit error may cause a wrong iteration of decryption in Wu and Deng scheme [12], resulting in wrong decryption of a whole CCP. These schemes have worse error resilience performance than ours. In Wu and Ma scheme [11], a bit error affects the byte where the error bit resides and possibly the next byte too. Similarly, a bit error affects locally for Fang and Sun scheme [14]. If a CCP contains multiple codeword segments and if the error bit is close to the boundary of two codeword segments and the error expansion due to encryption extends to the neighboring codeword segment, then the two schemes reported in [11, 14] have worse error resilience performance than ours. Otherwise, they have a similar error resilience performance as ours.

5. EXPERIMENTAL RESULTS

Our JPEG 2000 encryption scheme has been implemented on top of the open source JPEG 2000 C implementation JasPer [23] with the public domain C++ cryptographic library Crypto++ [24]. SEAL [25] was used as the stream cipher in CSE in our experiments. The default setting of JasPer was used in JPEG 2000 encoding unless stated otherwise. In the CSE experiments, that is, the experiments for Figures 6–8, the JPEG 2000 arithmetic codeword segment syntax was used, that is, any two consecutive bytes cannot be in the range from 0xFF90 to 0xFFFF, and the ending byte cannot be 0xFF.

Figure 6 shows the average number of bits for each disjoint sequence of switched bits, and Figure 7 shows the average number of disjoint sequences of switched bits for different lengths of plaintext with random syntax-compliant plaintexts as input to CSE. The average was over 15 000 runs. From Figures 6 and 7, we can see that ciphertext switching occurs about 8 times on the average for a plaintext of 5000 bytes and about 16 bits are switched back to the plaintext bits for each occurrence. That implies that there about 0.32% of the ciphertext bits is switched to the plaintext. Wu and Ma scheme [11] has more unencrypted bits than CSE on the average, and the locations of unencrypted bits are known. On the contrary, the locations of unencrypted bits in CSE are unknown and hard to locate if the JPEG 2000 bitstream is unknown. For both Wu and Deng scheme [12] and Fang and Sun scheme [14], all the plaintext bits are encrypted.

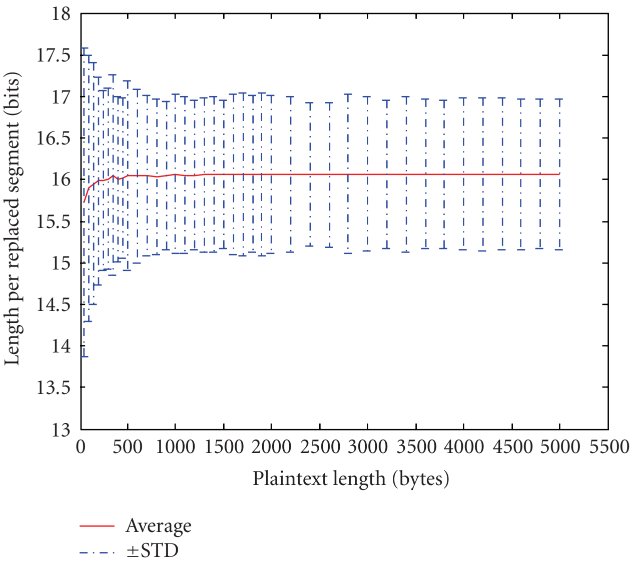

Figure 6: Average number of bits for each disjoint sequence of switched bits for JPEG 2000 arithmetic codeword segment syntax.

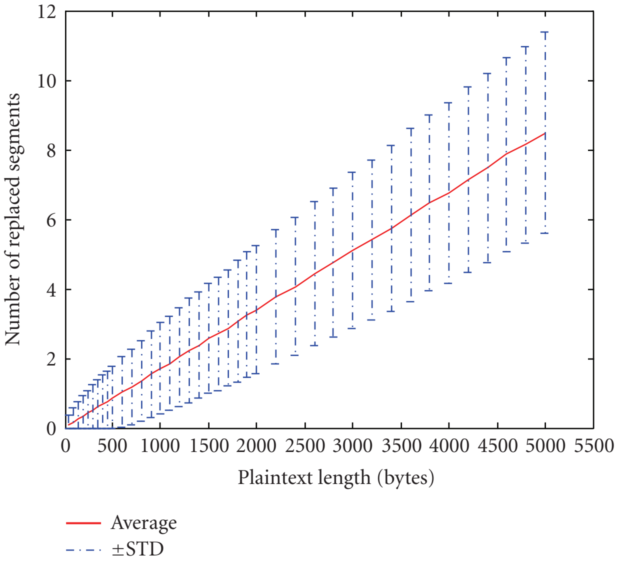

Figure 7: Average number of disjoint sequences of switched bits for different lengths of plaintext.

Since XOR operation is faster than modular addition or subtraction, and ciphertext switching occurs infrequently, we would expect that CSE is faster than other syntax-compliant encryption schemes. To confirm this, encryption and decryption speeds of CSE and other syntax-compliant encryption schemes were tested on a Dell OptiPlex GX280 PC with a 3.2 GHz Intel Pentium 4 CPU and 1 GB of RAM running Windows XP Professional Edition with SP2. Random syntax-compliant plaintexts of different lengths were used in the tests. Figure 8 shows the speeds averaged over 15 000 runs. Since encryption and decryption are identical for CSE, there

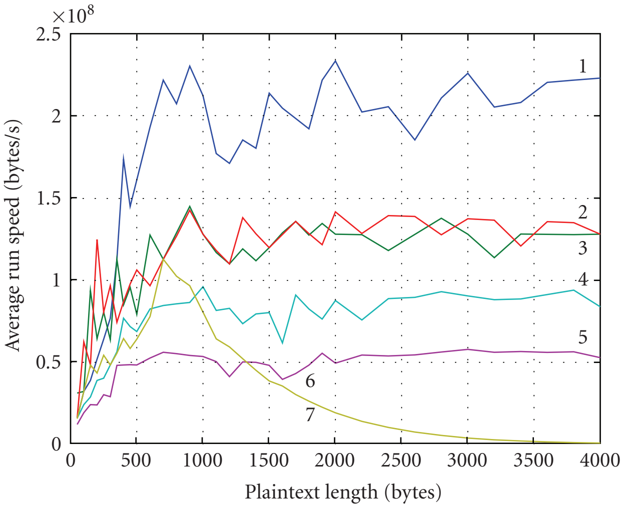

FIGURE 8: Encryption and decryption speeds averaged over 10 000 runs. (1) CSE. (2 and 3) Decryption and encryption of Fang and Sun scheme [14]. (4 and 5) Encryption and decryption of Wu and Ma scheme [11]. (6 and 7) Encryption and decryption of Wu and Deng scheme (using modular addition and subtraction) [12].

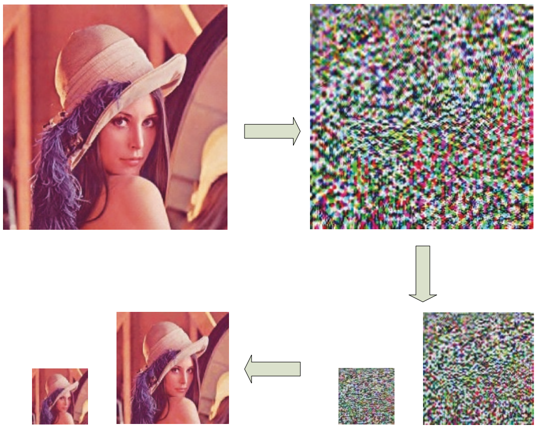

FIGURE 9: Syntax-compliant encryption that preserves full scalability of JPEG 2000.

is only a single speed curve for CSE in Figure 8. As shown in the figure, CSE is indeed faster than other schemes, especially when the plaintext length is larger than 500 bytes. The low speeds for small plaintext lengths are mainly due to the time spent on resetting IVs for SEAL, which accounts for a majority of the encryption time.

In Figure 9, we show that the original JPEG 2000 compressed image "Lena" is encrypted with the full scalability preserved. With our syntax-compliant encryption algorithm, the encrypted codestream can still be displayed by an encryption-unaware JPEG 2000 image viewer. The encrypted codestream is then truncated to lower resolutions. These truncated codestreams are still syntax-compliant and can be displayed by an encryption-unaware JPEG 2000 image viewer. They are then decrypted to recover the lower-resolution images.

TABLE 1: Speed overhead of our JPEG 2000 encryption scheme in the normal mode over Jasper's compression for different settings of layers and resolutions.

	Resolutions			
Layers (%)	5	4	3	2
10	0.7569	0.8959	1.213	1.033
8	0.3796	0.4411	0.5645	0.6193
6	0.2180	0.6315	0.5261	0.8368
4	0.5994	0.5778	1.214	0.6066
2	0.4190	0.4227	0.6189	0.6098

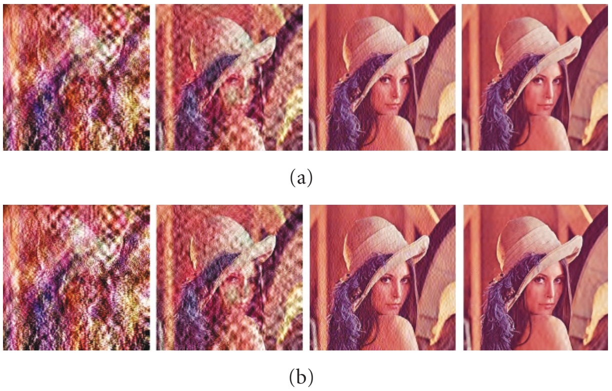
(a)
(b)

FIGURE 10: Rendered images for unencrypted (a) and encrypted (b) "Lena" for bit error rates of 0.00005, 0.0001, 0.001, and 0.01.

We have tested the file-size overhead of our encryption scheme. For all the testing images, our scheme has always 11 bytes more than the original JPEG 2000 compressed codestream, no matter how large the codestream is. These 11 additional bytes are in fact the SEC header we added to a codestream encrypted with our scheme. We conclude that our scheme has negligible file-size overhead.

We have also tested the speed overhead of our encryption in the normal mode over Jasper's compression with the standard color image "Mandrill" of size 512 × 512 on a Dell Inspiron 640 m laptop with a 1.6 GHz Intel CPU and 1.50 GB memory running Windows XP Professional SP2. A compression rate of 0.1 BPP was used. Our timing consists of the in-memory encoding and encryption part, with disk I/O excluded. The speed overhead for different settings of layers and resolutions are shown in Table 1. The maximum overhead is about 1.2%. We conclude that our JPEG 2000 encryption scheme has a negligible impact on the encoding speed.

To test error resilience of our encryption scheme, we have performed a blind perceptual test in which two rendered images were displayed side by side. One image was unencrypted and the other one was the corresponding encrypted image. Which image was encrypted was unknown to the viewers. No error resilience syntax or tools were used in JPEG 2000 encoding in our tests. Data bits were randomly selected and flipped at the same positions (if the added SEC header in the encrypted codestream was ignored) in both encrypted and unencrypted codestreams to simulate transmission bit errors. Eight standard test images were used in

Table 2: Blind perceptual test for error resilience.

Test image	Original better (%)	Encrypted better (%)
Barbara	63	57
Cameraman	52	68
House	62	58
Lake	54	66
Lena	55	65
Mandrill	63	57
Peppers	62	58
Walkbridge	53	67

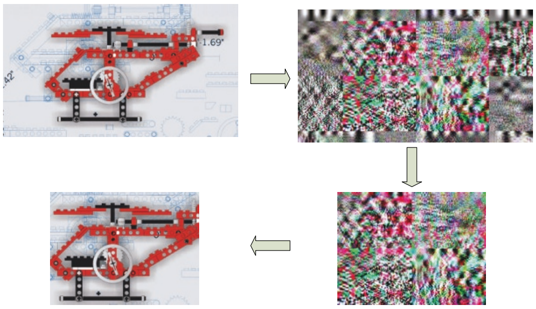

Figure 11: Cropping an encrypted JPEG 2000 image from aspect ratio 16 : 9 to 4 : 3 (e.g., 1280 × 720 to 792 × 594).

the blind perceptual test. Four bit error rates, that is, 0.01, 0.001, 0.0001, and 0.00005, were used. The rendered images for both encrypted and unencrypted "Lena" are shown in Figure 10 for the four bit error rates. Thirty volunteers, half with expertise in image processing and the other half does not, were asked to choose the one perceptually better from the two images displayed side by side. The testing result is summarized in Table 2. The difference is statistically insignificant. We conclude that our encryption does not degrade the perceptual quality as compared with the unencrypted case, and therefore our encryption scheme is error-resilient.

Figure 11 shows an encrypted JPEG 2000 image of aspect ratio 16 : 9 adapted to 4 : 3 by truncating some boundary tiles. Such aspect ratio adaption is widely used when DVD movie is displayed on a traditional TV. The original image size is 1280 × 720, the tile size is 396 × 234, the tile grid origin on the coordinate canvas is (0, 0), and the image origin is at (152, 234). With the above carefully chosen parameters, the encoded JPEG 2000 image has four tiles at the center, forming an area of aspect ratio 4 : 3. Adaptation from aspect ratio 16 : 9 to 4 : 3 is simply to truncate all the tiles except the four tiles at the center.

6. CONCLUSION

We have presented a novel syntax-compliant encryption primitive and an efficient syntax-compliant JPEG 2000 encryption scheme. The syntax-compliant encryption primitive, that is, the ciphertext switching encryption (CSE), produces syntax-compliant ciphertext with exactly the same length as the input. It is faster than all the other syntax-compliant encryption primitives. Our JPEG 2000 encryption scheme has two encryption modes. In the normal mode, each codeblock segment is independently encrypted. In the in situ mode, each intersection of a codeword segment and a codeblock contribution to a packet (CCP) is independently encrypted. A set of truncation-invariant parameters is used to uniquely identify each independently encrypted data block. These parameters are combined with a global initialization vector (IV) to generate on the fly the IV used to encrypt the data block these parameters refer to. A codestream encrypted with our scheme is syntax-compliant. The original granularity of scalability is fully preserved after encryption so that an encrypted codestream can be truncated to adapt to different representations without decryption. Our JPEG 2000 encryption scheme is fast, error-resilient, and has negligible file-size overhead.

REFERENCES

[1] ISO/IEC, "Information Technology—JPEG 2000 Image Coding System, Part 1: Core Coding System," ISO/IEC 15444-1:2000 (ISO/IEC JTC/SC 29/WG 1 N1646R), March 2000.

[2] ISO/IEC, "JPEG2000 verification model 8.5 (technical description)," ISO/IEC JTC 1/SC 29/WG 1 N1878, September 2000.

[3] ISO/IEC, "JPSEC commission draft 2.0," ISO/IEC/JTC 1/SC29/WG 1, N3397, 2004.

[4] F. Dufaux, S. Wee, J. Apostolopoulos, and T. Ebrahimi, "JPSEC for secure imaging in JPEG 2000," in *Applications of Digital Image Processing XXVII*, vol. 5558 of *Proceedings of SPIE*, pp. 319–330, Denver, Colo, USA, August 2004.

[5] B. B. Zhu, M. D. Swanson, and S. Li, "Encryption and authentication for scalable multimedia: current state of the art and challenges," in *Internet Multimedia Management Systems V*, vol. 5601 of *Proceedings of SPIE*, pp. 157–170, Philadelphia, Pa, USA, October 2004.

[6] B. B. Zhu, "Multimedia encryption," in *Multimedia Security Technologies for Digital Rights Management*, W. Zeng, H. Yu, and C.-Y. Lin, Eds., chapter 4, pp. 75–109, Elsevier, London, UK, 2006.

[7] P. P. Dang and P. M. Chau, "Image encryption for secure Internet multimedia applications," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 3, pp. 395–403, 2000.

[8] National Bureau of Standards, "Data Encryption Standard," NBS FIPS Pub. 46, January 1977.

[9] R. Grosbois, P. Gerbelot, and T. Ebrahimi, "Authentication and access control in the JPEG 2000 compressed domain," in *Applications of Digital Image Processing XXIV*, vol. 4472 of *Proceedings of SPIE*, pp. 95–104, San Diego, Calif, USA, July 2001.

[10] S. Wee and J. Apostolopoulos, "Secure scalable streaming and secure transcoding with JPEG-2000," in *Proceedings of IEEE International Conference on Image Processing (ICIP '03)*, vol. 1, pp. 205–208, Barcelona, Spain, September 2003.

[11] H. Wu and D. Ma, "Efficient and secure encryption schemes for JPEG2000," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '04)*, vol. 5, pp. 869–872, Montreal, Quebec, Canada, May 2004.

[12] Y. Wu and R. H. Deng, "Compliant encryption of JPEG2000 codestreams," in *Proceedings of the International Conference on Image Processing (ICIP '04)*, vol. 5, pp. 3439–3442, Singapore, October 2004.

[13] O. Watanabe, A. Nakazaki, and H. Kiya, “A scalable encryption method allowing backward compatibility with JPEG2000 images,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS ’05)*, vol. 6, pp. 6324–6327, Kobe, Japan, May 2005.

[14] J. Fang and J. Sun, “Compliant encryption scheme for JPEG 2000 image code streams,” *Journal of Electronic Imaging*, vol. 15, no. 4, Article ID 043013, 4 pages, 2006.

[15] B. B. Zhu, Y. Yang, and S. Li, “JPEG 2000 encryption enabling fine granularity scalability without decryption,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS ’05)*, vol. 6, pp. 6304–6307, Kobe, Japan, May 2005.

[16] B. B. Zhu, Y. Yang, and S. Li, “JPEG 2000 syntax-compliant encryption preserving full scalability,” in *Proceedings of IEEE International Conference on Image Processing (ICIP ’05)*, vol. 3, pp. 636–639, Genova, Italy, September 2005.

[17] B. B. Zhu, Y. Yang, C. W. Chen, and S. Li, “Fine granularity scalability encryption of MPEG-4 FGS bitstreams,” in *Proceedings of 7th IEEE Workshop on Multimedia Signal Processing (MMSP ’05)*, pp. 1–4, Shanghai, China, October 2005.

[18] D. S. Taubman, “High performance scalable image compression with EBCOT,” in *Proceedings of International Conference on Image Processing (ICIP ’99)*, vol. 3, pp. 344–348, Kobe, Japan, October 1999.

[19] D. S. Taubman, *JPEG2000 Image Compression: Fundamentals, Standards and Practice*, Kluwer Academic, Dordrecht, The Netherlands, 2001.

[20] T. Acharya and P. S. Tsai, *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*, John Wiley & Sons, New York, NY, USA, 2005.

[21] A C++ implementation of the Ciphertext Switching Encryption (CSE), http://research.microsoft.com/~binzhu/codes/CSE/.

[22] M. Bellare and P. Rogaway, “Introduction to Modern Cryptography,” chapter 4, Symmetric Encryption, http://www.cse.ucsd.edu/~mihir/cse207/classnotes.html.

[23] JasPer, http://www.ece.uvic.ca/~mdadams/jasper/.

[24] Crypto++, http://www.eskimo.com/~weidai/cryptlib.html.

[25] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, New York, NY, USA, 2nd edition, 1996.