2007 IEEE International Conference on Signal Processing and Communications (ICSPC 2007), 24-27 November 2007, Dubai, United Arab Emirates

EFFICIENT TRANSPARENT JPEG2000 ENCRYPTION WITH FORMAT-COMPLIANT HEADER PROTECTION

Dominik Engel, Thomas Stütz, and Andreas Uhl

Department of Computer Sciences
University of Salzburg
Salzburg, Austria
Email: {dengel,tstuetz,uhl}@cosy.sbg.ac.at

ABSTRACT

Transparent encryption allows the decoding of a preview image from the encrypted bitstream, even if the key is not available. We propose a lightweight partial encryption scheme that allows transparent encryption for JPEG2000. The proposed scheme efficiently encrypts JPEG2000 packet header information. The resulting bitstream is fully format-compliant. Apart from low computational complexity, the proposed scheme has several beneficial properties, e.g. it allows to perform tasks such as rate-adaptation and visual hashing in the encrypted domain. We evaluate the security of the proposed approach and discuss possible attacks. The proposed approach succeeds in protecting the full-quality version of the visual content reasonably well, especially considering its low computational demands.

Index Terms— JPEG2000, format-compliance, packet header encryption, transparent encryption, lightweight encryption

1. INTRODUCTION

Scalability in the representation of visual data is quickly becoming more important, as the requirement of serving many different terminal devices with different display capabilities (especially regarding resolution, but also computational capacity) with a single bitstream needs to be satisfied. Since the JPEG2000 standard has been passed, the format has not exactly taken the field by storm, but because JPEG2000 provides scalability and superior compression performance compared to JPEG and other formats, it is to be expected that the use of JPEG2000 will increase significantly in the future. With a more widespread use of JPEG2000, different security techniques that cover the range of applications need to be devised.

"Transparent encryption" denotes encryption schemes for which a preview image can be decoded from the encrypted stream, even without the key data [1]. Broadcasting applications, for example, benefit from transparent encryption, as they, rather than preventing unauthorized viewers from receiving and watching their content completely, often aim at promoting a contract with non-paying watchers, for whom the availability of a preview version (in lower quality) may serve as an incentive to pay for the full quality version. The straightforward approach to achieve transparency is to encrypt packet data at the end of the bitstream, which corresponds to encryption of enhancement layers [2]. Since the major part of the packet body data needs to be encrypted, this approach is demanding in terms of computational complexity.

Another aspect of encryption schemes is whether they offer format-compliance. If a scheme is fully format-compliant, then the encrypted bitstream can be interpreted by any JPEG2000 compliant decoder. Tasks such as rate-adaption can be performed in the encrypted domain. There is a number of approaches that propose format-compliant encryption for JPEG2000, e.g. [3, 4, 5], all of which focus on the encryption of packet bodies. In this paper we discuss an efficient method for fully format-compliant transparent encryption that focusses on the packet headers: only the packet headers are encrypted, the packet body data remain in plaintext.

2. FORMAT-COMPLIANT HEADER ENCRYPTION

The information contained in the JPEG2000 packet header relates to inclusion of codeblocks in the packet, the length of each codeblock contribution to the packet (CCP), the number of coding passes and the number of leading zero bitplanes [6]. If this information is missing, then a reconstruction will yield a distorted image, as the packet body data is misinterpreted.

In a format-compliant encryption approach for packet headers, the encrypted header information should be consistent with the available packet body data. Furthermore, it should of course be possible to reconstruct the original packet headers from the encrypted packet header data by the use of a key. In the following subsections, we propose a reversible, key-dependent transformation for each piece of information in the packet header. Thereby the key is used to create a (pseudo-)random keystream. The choice of random generator can range from a physical source of randomness to using a symmetric cipher like AES in output feedback mode. In our tests we use a linear congruential random generator.

As each of the transformations can be limited to a single packet, the quality of the preview image can be controlled relatively precisely. The headers of the packets (at the beginning of the bitstream) that should constitute the preview image are simply left in plaintext. The decoder needs to be signalled the amount of data it is allowed to decode in order to obtain the preview image.

2.1. CCP Lengths and Number of Coding Passes

JPEG2000 explicitly signals both, the number of coding passes and the length of each codeblock contribution [6]. In order to prevent access to this information, we distribute the CCP lengths and number of coding passes in each packet among all non-empty codeblocks. The number of coding passes gives only little information to a potential attacker, the lengths of the codeblock

This work has been partially supported by the Austrian Science Fund (FWF) under Project No. P19159-N13 and by the European Commission through the IST Programme under contract IST-2002-507932 ECRYPT.

contributions are a more valuable source of information, as they are more distinctive. The transformation we propose here can be applied to both (for clarity of reading we give the description for lengths only).

The requirements are rather unique for this transformation: given a vector of non-zero positive integers (the CCP-lengths or the number of coding passes) and a random keystream, we want an output vector that randomly redistributes the lengths among all positions (using all possible mappings), preserves the overall sum of the lengths, and that has the same number of elements, each of which has to be a non-zero positive integer. Furthermore, the transformation needs to be reversible: given the random keystream and the output vector, the original vector has to be reconstructible.

To achieve a transformation that adheres to these requirements, we change packet lengths (and number of coding passes, respectively) in overlapping pairs, starting with the first and the second length, moving on to the second and the third length, and so on. We redistribute the lengths between a pair of lengths by adding a random number from 0 to the total length of the two packets. This addition is performed modulo the packet size minus one and after the modulo operation we add one. Thus the size of any packet can never become zero. To avoid that an attacker can obtain the original sum of the pairs, we shuffle the lengths before and after redistributing them.

The number of possible alterations depends on the number of codeblocks, and how they make their contributions to the individual packets. If a small number of packets contains many contributions more alterations are possible than if a large number of packets only contain a small number of contributions each. Two non-empty codeblock contributions in a packet are enough to hide their lengths and number of coding passes. Therefore, in practical scenarios, there will always be ample opportunity to sufficiently randomize CCP lengths and number of coding passes.

2.2. Leading Zero Bitplanes

The number of leading zero bitplanes (lzb) for each codeblock is coded by using tag trees [6]. When the number of lzb is changed, the length of the output from the associated tag tree might change as well. This change in length has to be reflected in the tile header, otherwise the decoder will complain.

To transform the number of leading zero bitplanes we simply use the keystream to generate random bytes. We then add a random byte to the number of leading zero bitplanes modulo a previously determined maximum number of skipped bitplanes. Note that the new number of skipped bitplanes needs to be greater or equal to the original number. As the maximum number of bitplanes can be derived from information contained in the main header [7], the main header needs to be changed for full-format compliance (otherwise decoding will still work, but the decoder might issue a warning). For decoding, the random byte is subtracted instead of added.

The number of possible alterations obviously depends on the codeblock size. For smaller sizes, more codeblocks exist, and more alterations of the lzbs can be performed.

2.3. Inclusion Information

Each packet contains the inclusion information for a certain quality layer for all codeblocks in the resolution (for the sake of simplicity, we assume that no precinct partitioning is used). There are four types of inclusions that codeblock c can have in packet p: *FI* (c is included in p for the first time, i.e. c has not been included in any previous packet), *NI* (c is not included in p and has never been included in any previous packet), *PI* (c has been included in a previous packet and is also included in p), and *PN* (c has been included in a previous packet but is not included in p). The sequence of inclusion information of each codeblock is coded depending on the type of inclusion. *FI* and *NI* are coded by an inclusion tag tree. For *PI* and *PN*, i.e. previous inclusions, a 1 is coded in the header if the codeblock is included again in the current packet, and a 0 is coded if it is not included in the current packet.

We distinguish two kinds of packets: an *empty packet* is a packet for which a header is written, but which does not contain any codeblock contributions, i.e. all codeblocks are included as *NI* or *PN*. In a *non-empty packet* there is at least one contribution from one of the codeblocks, i.e. at least one codeblock is included as *FI* or *PI*. We define the *inclusion vector* v_p for a packet p as the sequence of the inclusion information for each codeblock associated with p. The order of elements in the inclusion vector follows the order in which the codeblocks are scanned during image coding. In JPEG2000 this is an ordering by subband in the sequence HL, LH, HH followed by a lexicographical ordering over the 2-D coordinates of all codeblocks in the subband. Obviously, the order in which subsequent items of the same inclusion type appear are irrelevant, and count as the same permutation.

In order to produce a format-compliant bitstream, after the first permutation, the permutations for the subsequent packets have to regard the inclusion information that has been signalled in the directly preceding packet. The following transitions are possible:

p_l	*FI*	*NI*	*PI*	*PN*
p_{l-1}	*NI*	*NI*	*FI*, *PI*, *PN*	*FI*, *PI*, *PN*

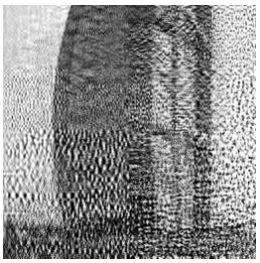
A codeblock can only be included as *FI* or *NI* in p_l if it has been included as *NI* in packet p_{l-1}. *PI* and *PN* can follow a previous inclusion of *FI*, *PI*, or *PN*. It follows that permutations can only be performed for non-empty packets, because for empty packets the positions of the inclusions of types *NI* and *PN* are fully determined by the previous packet.

Initially, in each resolution, the first packet suitable for permutation is searched: This packet has to have at least one non-empty codeblock contribution (*FI* or *PI*) and one empty inclusion (*NI* or *PN*). Then, for the subsequent non-empty packets, the inclusion information is permuted regarding the possible transitions. The inclusion information in empty packets cannot be permuted and is only updated according to the previous packet. This procedure results in a (semantically) consistent sequence of inclusion information for the packets of each resolution. The number of possible permutations for a given input image depends primarily on the number of codeblocks, which in turn depends on the used compression settings. For example, for the test image shown in figure 1(a) with a bitrate of 1bpp, a codeblock size of 64×64, and 32 quality layers, 2^{987} permutations are possible.

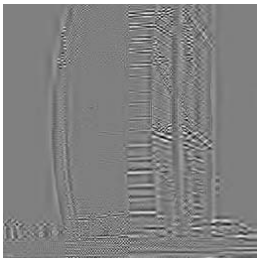
2.4. Combined Format-Compliant Header Transformation

The format-compliant transformation of the different pieces of information in the packet headers can and should be combined. The order in which they are applied is arbitrary, only decoding has to apply the reverse transformations in the reverse order. The format compliance of the combined format-compliant header encryption has been verified experimentally by decoding the encrypted files with the reference implementations JasPer and JJ2000. The effectiveness of the transformations depends on the settings of the compression parameters, as discussed above. For small codeblock sizes and many quality layers, the keyspace size is increased dramatically. If packet boundaries were crossed and furthermore inclusion information was split up among codeblocks, the keyspace could be further increased. However, the

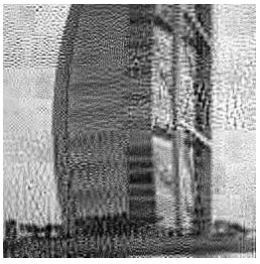
(a) Test image



(b) CCP lengths (& number of coding passes), PSNR 9.7 dB

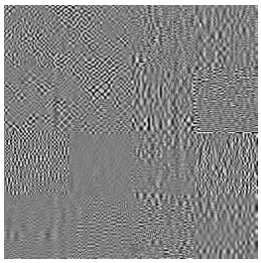


(c) Number of lzb, PSNR 10.6 dB

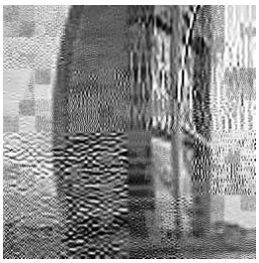


(d) Inclusion information, PSNR 12.1 dB

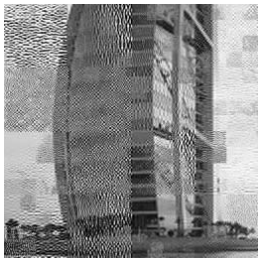
Figure 1. Test image & visual examples of reconstructions with transformed packet headers: 1 bpp, cblk. size 32×32, 32 layers



(a) Transformations start at resolution 0, PSNR 7.2 dB



(b) Transformations start at resolution 2, PSNR 9.7 dB



(c) Transformations start at resolution 3, PSNR 9.8 dB

(d) Preview image for 2(c) (yields resolution 2), PSNR 10.2 dB

Figure 2. Visual examples of reconstructions for all transformations and different levels of transparency

downside would be a loss in semantics for the encrypted version (which would make rate adaption more unreliable, for example).

2.5. Visual Examples

In order to give an illustration of the impact of encrypting each of the pieces of header information, we give some visual examples. The test image (courtesy of Jumeirah) is shown in figure 1(a). It is an 8bpp grayscale image of 1024×1024 pixels. For the illustration we used a bitrate of 1 bpp, a codeblock size of 32×32 and 32 quality layers.

Figures 1(b) to 1(d) show direct reconstructions of the image with a single piece of header information encrypted (with header transformation for the encrypted image starting at the first resolution). The impact of transforming the different parts of the header information can be observed. Figures 2(a) to 2(c) show the direct reconstruction for all header transformations combined, with the start of the transformation set to a different resolution level for each figure. It can be seen that if the transformations start at a higher resolution level, lesser distortion is introduced. It should be noted that the direct reconstructions do not represent the preview images, but rather the result of reconstructing *all* of the available (transformed) data (in a way, it is the attacker's view). In order to get a preview image, the portion of the bitstream that has not been distorted needs to be decoded. An actual preview image is shown in Figure 2(d). This preview image was taken from a bitstream where the transformations start at resolution 3, so the preview image is equal to resolution 2 of the original image (i.e. the third resolution contained in the bitstream).

3. EFFICIENCY

The transformations used in the proposed scheme are computationally cheap. The array-based permutations used in the transformations are all of linear complexity. The necessary operations in each substep are the generation of a random number and exchanging the places of two items in the array. The size of the packet headers compared to the packet bodies depend on the compression settings. As an example, we take the test image shown in figure 1(a) of 1024×1024 pixels at a compression rate of 1 bpp. The following table shows the number of header and body bytes and the ratio of header bytes to the total number of bytes in the bitstream.

Cblk. Size	Layers	Header Bytes	Body Bytes	Ratio
64×64	16	1823	129072	1.4%
32×32	32	4603	126283	3.5%
16×16	32	11379	119510	8.5%
8×8	32	25748	105176	18.6%

As can be seen, the size of the headers increases with the number of codeblocks and the number of layers. With more information in the headers to be permuted, the scheme needs more computational complexity, but also gives more security, as the number of possible transformations increases. The proposed scheme is significantly more efficient than the straightforward approach that needs to encrypt the major part of the packet data.

4. SECURITY

The complexity of a brute-force attack that tries to obtain the full quality version of the image depends on the number of possible transformations. For practical settings the obtained number of possible combined transformation will be very high (2^{987} for the inclusion information alone in the example used above). However, the complexity of a brute-force attack can only give an upper-bound for security, often there will be easier attacks. This is especially true for partial encryption [8].

The proposed key-dependent transformations use permutations, which are principally vulnerable to known-plaintext attacks. Another security threat for the proposed scheme is the fact that an attacker can iteratively try to get a better quality than allowed. The attacker can make a guess on the header information (using the information that is preserved by the encryption, e.g. the types of inclusions in the packet), and then iteratively perform systematic alterations on the header information and reconstruct an image. In each iteration, if the quality improves compared to the previous construction, the alteration is accepted, otherwise the previous state is restored.

The problem for the attacker is that there is no measure with which to assess the quality of the obtained image (apart from using a human observer). The demands for an attack increase with the number of resolutions. The lower resolutions will have significantly fewer packets than the higher resolutions, therefore an attack will be easier (and possibly be performable by a human observer) on the lower resolutions, while the protection of the higher resolutions is stronger. However, it should be taken into account that obtaining a version of better visual quality than the preview image is possible without too much effort, and (compression) settings should be chosen accordingly.

Full confidentiality, i.e. no preview image at all and high security for the visual data, cannot be provided by the proposed scheme, even if the packet body data for the lowest resolution is encrypted. This is because the protection for the lower resolutions is significantly weaker than the protection of the higher resolutions, and it will almost always be possible for an attacker to get a rough estimate of the visual content from these lower resolutions.

5. APPLICABILITY

Apart from the advantages that are normally gained with format-compliant encryption, the fact that the packet body data remains in plaintext also enables techniques that can be used for indexing, retrieval and classification. For example, visual hashing [9] can be applied without the need to decrypt. Generally, applications that use the packet body data for classification, indexing and retrieval will work with the proposed scheme. Also, schemes that use general information from the header will work. For example, the scheme proposed by [10] uses the number of bytes spent on each subband as a texture classification tool. Although this data is retrieved from the header, this scheme will still work with encrypted headers, as the sum of all codeblock contributions is preserved by the proposed encryption scheme. The possibility to perform these tasks in the encrypted domain can be an important feature, which the packet-body-based techniques cannot provide.

On the other hand, classification, indexing and retrieval techniques based on detailed features of the packet header can be disabled by the proposed scheme. For example, the approach by [11], which uses the number of leading zero bitplanes as a fingerprint, can be disabled, just as the approach by [12] which uses a set of classifiers based on the packet header and packet body data.

6. CONCLUSION

The proposed transparent encryption scheme offers two advantages: efficiency and format-compliance. As only the header information needs to be encrypted, the proposed scheme has low computational demands, even compared to other partial encryption schemes. Full format-compliance allows to perform various tasks in the encrypted domain, including indexing, classification and retrieval techniques. The quality of a preview image for transparent encryption can be controlled quite precisely by choosing an appropriate point in the sequence of packets where to start header transformation.

As regards security, the scheme definitely and expressly has to be denoted lightweight. The use of permutations makes it vulnerable to known-plaintext attacks, iterative attacks can obtain higher quality images than the preview image. However, the full quality version remains reasonably secure, considering the low effort used for encryption.

7. REFERENCES

[1] B. M. Macq and J.-J. Quisquater, "Cryptology for digital TV broadcasting," *Proceedings of the IEEE*, vol. 83, pp. 944–957, June 1995.

[2] A. Uhl and C. Obermair, "Transparent encryption of JPEG2000 bitstreams," in *Proceedings EC-SIP-M 2005 (5th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services)* (P. Podhradsky *et al.*, eds.), (Smolenice, Slovak Republic), pp. 322–327, 2005.

[3] T. Stütz and A. Uhl, "On format-compliant iterative encryption of JPEG2000," in *Proceedings of the Eighth IEEE International Symposium on Multimedia (ISM'06)*, (Los Alamitos, CA, USA), pp. 985–990, IEEE Computer Society, 2006.

[4] H. Kiya, D. Imaizumi, and O. Watanabe, "Partial-scrambling of image encoded using JPEG2000 without generating marker codes," in *Proceedings of the IEEE International Conference on Image Processing (ICIP'03)*, vol. III, (Barcelona, Spain), pp. 205–208, Sept. 2003.

[5] Y. Wu and R. H. Deng, "Compliant encryption of JPEG2000 codestreams," in *Proceedings of the IEEE International Conference on Image Processing (ICIP'04)*, (Singapure), IEEE Signal Processing Society, Oct. 2004.

[6] D. Taubman and M. Marcellin, *JPEG2000 — Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, 2002.

[7] ISO/EIC 15444-1, "Information technology – JPEG2000 image coding system, Part 1: Core coding system," Dec. 2000.

[8] A. Said, "Measuring the strength of partial encryption schemes," in *Proceedings of the IEEE International Conference on Image Processing (ICIP'05)*, vol. 2, Sept. 2005.

[9] R. Norcen and A. Uhl, "Robust visual hashing using JPEG2000," in *Eighth IFIP TC6/TC11 Conference on Communications and Multimedia Security (CMS'04)* (D. Chadwick and B. Preneel, eds.), (Lake Windermere, GB), pp. 223–236, Springer-Verlag, Sept. 2004.

[10] A. Tabesh, A. Bilgin, K. Krishnan, and M. W. Marcellin, "JPEG2000 and motion JPEG2000 content analysis using codestream length information," in *Proc. Data Compression Conference, DCC 2005*, pp. 329–337, IEEE Computer Society Press, Mar. 2005.

[11] C. Liu and M. Mandal, "Fast image indexing based on JPEG2000 packet header," in *Proceedings of the 2001 ACM workshops on Multimedia: Multimedia Information Retrieval*, (New York, NY, USA), pp. 46–49, ACM Press, 2001.

[12] A. Descampe, P. Vandergheynst, C. D. Vleeschouwer, and B. Macq, "Coarse-to-fine textures retrieval in the JPEG 2000 compressed domain for fast browsing of large image databases," in *Proc. Multimedia Content Representation, Classification and Security, MRCS 2006* (B. Günsel, A. K. Jain, A. M. Tekalp, and B. Sankur, eds.), vol. 4105 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg, New York, Tokyo), pp. 282–289, Springer-Verlag, Sept. 2006.