Format-Compliant JPEG2000 Encryption with Combined Packet Header and Packet Body Protection[*]

Dominik Engel, Thomas Stütz, and Andreas Uhl
Department of Computer Sciences
University of Salzburg
Salzburg, Austria
{dengel,tstuetz,uhl}@cosy.sbg.ac.at

ABSTRACT

All proposals for format-compliant encryption schemes for JPEG2000 that have been made to date only encrypt packet body data, but leave packet header data in plaintext. In this paper we show that for providing strict confidentiality, leaving the packet header in plaintext severely compromises the security of these schemes, as discriminative – and for some settings even visual – information can be extracted from the header. We propose a set of format-compliant transformations of the packet header data that confines this information leakage. Furthermore, we discuss to what extent the proposed header protection scheme may be used to increase the performance of partial / selective encryption schemes that, rather than providing full confidentiality, trade off security for a decrease in computational demands.

Categories and Subject Descriptors

I.4.2 [**Image Processing and Computer Vision**]: Compression (Coding); E.3 [**Data**]: Data Encryption

General Terms

Security

Keywords

JPEG2000, packet header encryption, packet body encryption, format-compliant encryption

1. INTRODUCTION

As scalable representations of visual data are steadily becoming more widespread due to the necessity to serve the different needs of the increasing variety of commonly used display devices – each with distinct capabilities, e.g., in terms of display resolution and computing power –, the development of security techniques that are tailored to the specific requirements of scalable representations becomes crucial. In the area of still image coding, JPEG2000 [18] is the most recent and advanced standard in scalable representation of visual data. A large number of propositions has been made concerning the security of JPEG2000-coded visual data. One of them is the very recent addition to the JPEG2000 standard in form of part 8, JPSEC [8].

While the straightforward approach to encrypt the whole JPEG2000 bitstream (note that our usage of the term "bitstream" differs from the nomenclature in the JPEG2000 standard, which refers to the JPEG2000 bitstream as codestream) with a classical, strong cryptographic cipher like AES remains the most secure option, there are several reasons why alternative approaches have been discussed. Two of the main goals are (a) a decrease in computational demands and (b) the preservation of JPEG2000 functionality. Some approaches aim at providing both (while still trying to maintain a high level of security).

There are two ways in which (a) can be achieved. One way is to employ cryptographic primitives that are less demanding (but also less secure). This is usually called "soft" encryption. Another way is to selectively encrypt only vital parts of the bitstream with a secure classical cipher. This is termed "partial / selective" encryption. Approaches trying to satisfy (b) seek to transfer some of the functionality of a JPEG2000 bitstream to the encrypted bitstream. Full encryption does not take into account the features JPEG2000 offers, especially with respect to scalability: because the resulting bitstream cannot be interpreted by the decoder, all functionality is destroyed. Approaches which produce an encrypted bitstream that can be interpreted by the decoder are termed "format-compliant". The security of format-compliant encryption schemes is the primary concern of this paper. Note that by a format-compliant bitstream we refer to a bitstream that can be interpreted by *any* JPEG2000 decoder, i.e., the bitstream is compliant with part 1 of the JPEG2000 standard (cf. part 4 of the JPEG2000 standard [7]).

A number of approaches have been proposed, which aim at achieving fully format-compliant encryption (sometimes in conjunction with a reduction in computational demands), e.g., [5, 22, 14, 1, 9, 4, 13, 23, 21, 2, 16]. Segment-based encryption as proposed by [20, 19] does not fulfill the requirement for full format-compliance in our sense, as in this case rate-adaption can only be performed by a JPSEC-compliant decoder. All of these approaches, including the method

[*]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.

sketched in FCD-15444-8[1], and to the best of our knowledge all related approaches that have been put forward to date, propose the exclusive encryption of the packet *body* data. The packet *header* data are invariably left in plaintext.

In Section 2 of this paper, we will discuss to what extent leaving the packet header data in plaintext is problematic in terms of security for approaches that aim at providing full confidentiality. (As opposed to approaches that only offer a degradation in visual quality, these approaches aim at actually fulfilling the same function as full encryption and not disclosing any of the visual content.)

In Section 3, we propose a set of transformations that allows to protect the JPEG2000 packet header data in a fully format-compliant way, i.e., the resulting bitstream can be processed like a "normal" JPEG2000 bitstream, by any decoder compliant with JPEG2000, part 1.

In Section 4, we first discuss general security considerations and then evaluate the proposed scheme with respect to two scenarios. First, in Section 4.1, we look at how header protection can confine the leakage of information from the packet header for encryption schemes that aim at providing full confidentiality in a format-compliant way. Second, approaches that aim at format-compliant encryption that provides sufficient degradation in quality, rather than full confidentiality (and trades off security for a decrease in computational demands), can be improved by a combination with the proposed header transformation. We discuss the possibilities and limitations of the proposed scheme with respect to this second scenario in Section 4.2. Section 5 concludes.

2. JPEG2000 HEADER INFORMATION

The main JPEG2000 header contains general information on the bitstream, e.g., the coding style settings. The information is too general to infer from it anything substantial with regard to the specific visual content. The same is true for the tile and tile-part headers.

The situation is different for the packet headers. The packet headers contain information on four properties of the packet. We will refer to each different kind of header information as a "class" in the following. The four classes are:

- the length of the contribution of each codeblock to the packet (CCP),
- the number of leading zero-bitplanes (LZB),
- the inclusion information of each codeblock,
- and the number of coding passes that are contained in the packet for each codeblock.

From the sum of the CCP lengths, the length of the packet body can be derived.

The packet header information is specific to the visual content, and it is specific enough to be used as a fingerprint. Some suggestions have been made in this direction in the context of indexing, retrieval and classification. [17] use the number of bytes spent on each subband for texture classification. [3] use a set of classifiers based on the packet header and packet body data to retrieve specified textures from JPEG2000 image databases. [10] use the number of leading bitplanes as a fingerprint to retrieve specific images. Any class of header information alone is discriminative enough

[1]http://www.jpeg.org/public/fcd15444-8.pdf

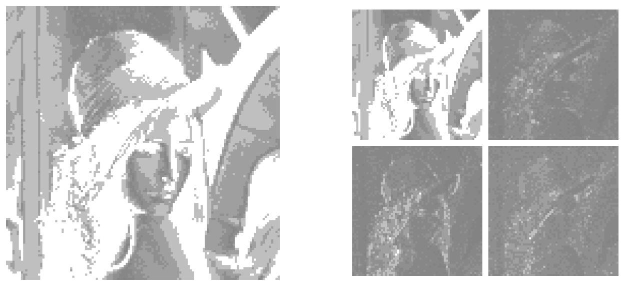

(a) wlev=0 (b) wlev=1

Figure 1: LZB images, codeblock size 4×4

to be potentially used as a fingerprint (as will be illustrated in Section 4.1). The combination of all classes definitely is.

In the context of a security application, the fact that the information contained in the packet headers can be used as a fingerprint for a specific image is quite problematic, as it means that an attacker can link an encrypted image to its plaintext. The more distinctive the fingerprint, the more accurate is this assignment.

Such a fingerprint alone is problematic enough. The situation is even worse with respect to the leading zero bitplanes. While the other classes of header information do not reveal any *visual* information, the number of leading zero bitplanes can be used to get a rough estimate of the original image from which, especially for small codeblock sizes, the visual content is discernible. Figure 1 shows images that can be created from the information on the number of leading zero bitplanes for the Lena image coded with a small codeblock size of 4×4. The LZB images are created by interpreting the number of leading zero-bitplanes as pixel values (normalized over the range of grayscale values). Figure 1(a) shows the LZB image that can be created if no wavelet transform is employed. It can be seen that the LZB image is a crude quantization on a codeblock basis, from which the basic visual content is easily discernible. Coding settings that employ no wavelet transform are not very likely to be employed in practice, but also if the image is wavelet transformed, the edge information in the highpass subbands (which are not further processed regardless of the decomposition depth) that is revealed by the LZB-information is enough to spoil full confidentiality, as shown in Figure 1(b). For larger codeblock sizes less information is leaked through the number of leading zero bitplanes, but in order to provide full confidentiality, this data should generally not be accessible.

3. FORMAT-COMPLIANT HEADER PROTECTION

In the format-compliant encryption scheme we propose here, on the one hand, the aim is to destroy the information needed to create a strong fingerprint. On the other hand, the information needed to perform such tasks as rate adaption should be kept intact: the distinction between packet body and packet header and the distinction between individual packets has to be made available to the decoder. The resulting bitstream also needs to be fully format-compliant, i.e., the transformed header information has to be consistent with the available (encrypted) 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 class 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 the standard (linear congruential) random generator provided by Java 2 Standard Edition 5.0.

3.1 CCP Lengths and Number of Coding Passes

JPEG2000 explicitly signals both, the number of coding passes and the length of each codeblock contribution. (Depending on the block coding, in other codecs these numbers can be derived from each other, in JPEG2000 they are independent, see [18]). 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 contributions are a more valuable source of information, as they are more distinctive. The algorithm we propose here can be applied to both (for clarity of reading we give the description for lengths only).

We need an approach that is key-dependent and reversible. Adding a random number modulo the total length of all CCPs to the offset of each CCP in the packet is not feasible: some offsets might change their relative positions during this procedure and the decoder would lack information about which random number is associated with which CCP. The situation leads to rather unique requirements for the 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 these number among all positions (using all possible mappings), preserves the overall sum of the original vector, and has the same number of elements, each of which has to be a non-zero positive integer.

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. We give the procedure in pseudo-code below. `v[]` is a vector of non-zero positive integers (indexing starts at 1). `random()` returns a random float number in $[0, 1)$. `mod` is the modulo operation, which can return a negative residual (as is the case in many programming languages).

```
shuffle (v)
borders := size(v)-1
for i := 1 to borders
  sum := v[i] + v[i+1]
  r := (int) random()*sum
  newBorder := ((v[i]+r) mod (sum-1)) + 1
  v[i] := newBorder
  v[i+1] := sum - newBorder
end for
shuffle(v)
```

The transformation can be reversed easily by unshuffling the input, traversing it from end to start, using the random numbers in reverse order, setting `newBorder` as:

```
newBorder := (v[1]-r-1) mod (sum-1)
if (newBorder <= 0) then
  newBorder := newBorder + (sum-1)
end if
```

and finally unshuffling the result again.

This approach allows to completely redistribute lengths and coding passes among the codeblocks in a packet. 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 numbers of coding passes.

3.2 Leading Zero Bitplanes

The number of leading zero bitplanes (LZB) for each codeblock is coded by using tag trees [18]. As discussed above, this information is even more critical than the other classes of header information, as by using the number of LZB an attacker can obtain information on the visual content of the encrypted image (for small codeblock sizes).

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. For decoding, the random byte is subtracted instead of added. The maximum number of skipped bitplanes needs to be signalled to the decoder, e.g., by inserting it into the key or by prior arrangement. Note that the maximum number of skipped bitplanes needs to be greater or equal to the original maximum number of skipped bitplanes (otherwise the modulo operation cannot be reversed). Theoretically the new number could be arbitrarily high (we found no restrictions in that respect in the standard), but most implementations will have a maximum number of bitplanes for the representation of coefficient data that must not be exceeded.

When the number of leading zero bitplanes 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. Alternatively, if only a single tile is used, the length in the tile header can be set to be unspecified. Furthermore, the maximum number of bitplanes needed to represent the coefficients in each subband can be derived from information contained in the main header: "The maximum number of bit-planes available for the representation of coefficients in any subband, b, is given by M_b as defined in Equation E.2" ([6], p. 70). Equation E.2 in [6] basically derives the number M_b from information contained in the `QCD` and `QCC` marker segments in the main header. Therefore, to achieve full format-compliance, the main header needs to be changed accordingly. Otherwise decoding will still work, but the decoder might issue a warning. Note, however, that neither of the reference implementations JJ2000 and JasPer, which we used in our tests, issued a warning.

The total number of possible changes for all packets depends on the number of available codeblocks. If more codeblocks exist, more information can be randomized.

3.3 Inclusion Information

Each packet contains the inclusion information for a certain quality layer for all codeblocks in the precinct associated with the packet. For the sake of simplicity, we assume that no precinct partitioning is used. In this case each packet contains inclusion information for all codeblocks of all subbands in the resolution associated with the packet. There are four types of inclusion 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., the codeblock has been included in a preceding packet, a 1 is coded in the header if the codeblock is included again in the current packet, and a 0 is coded if the codeblock is not included in the current packet.

The goal of the proposed approach is to permute inclusion information for each packet in such a way that the original inclusion information cannot be derived without the key and that the resulting "faked" inclusion information complies with the semantics of JPEG2000. We limit the approach to permutation of the original inclusion information and do not split available packet body data from one codeblock to more codeblocks, and we also do not merge contributions from distinct codeblocks in a single codeblock. Also, the permutation is applied per packet, i.e., we do not merge the packet body data from different packets, as this would have a detrimental effect on the scalability properties of the resulting bitstream. These restrictions do not interfere with the aim to prevent the creation of a strong fingerprint from the sequence of inclusions (although they would help to hide the number of inclusions of each type) and have the advantage of facilitating straightforward reversibility.

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 (i.e., each codeblock in the subbands of the resolution associated with p):

$$v = [I_{c_i}], \forall c_i \in p, I_{c_i} \in \{FI, NI, PI, PN\}.$$

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. We give the number of possible distinct permutations further below.

An arbitrary permutation of the inclusion vector of each packet would not produce a format-compliant bitstream. Consider the following example: After the permutation of the inclusion vectors for two packets p_l and p_{l+1} let codeblock c be included in packet p_l as *FI*. An arbitrary permutation could assign inclusion type *NI* to c in p_{l+1}. This would lead to a contradiction because c can never be *NI* after its first inclusion.

Only the first permutation in a resolution r may be an unrestricted permutation (but permutations do not necessarily have to start at the first packet of the first layer). After the first permutation, the permutations for the subsequent packets have to regard the inclusion information that has been signaled 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. Also note that the number of inclusions of type *FI* plus the number of inclusion of type *NI* in p_l is always equal to the number of inclusions of type *NI* in p_{l-1}. The same is true for inclusions of type *PI* and *PN*: their number in p_l is equal to the number of inclusions of type *FI*, *PI*, and *PN* in p_{l-1}.

We perform the permutations in compliance with these transition rules to produce a – syntactically as well as semantically – consistent sequence of inclusion information over the packets of each resolution. First, 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*). Packets for which the codeblocks all have the same inclusion information are obviously not suitable for permutation, and packets with mixed *FI* and *PI* only occur after the first candidate packet. The inclusion information in the first candidate packet is permuted.

For the subsequent non-empty packets, the inclusion information of the immediately preceding packets is regarded in the permutation. The inclusion vector v for a packet p_l is split into two vectors: $v^{(1)}$ contains all *FI* and *NI* inclusions, and $v^{(2)}$ contains all inclusions of type *PI* and *PN*. Both vectors are permuted randomly (using the key-stream) to form $\hat{v}^{(1)}$ and $\hat{v}^{(2)}$. According to the possible transitions given above, the elements of $\hat{v}^{(1)}$ are assigned (in the randomized order) to the positions that are marked as *NI* in the packet of the previous layer, p_{l-1}. The elements of $\hat{v}^{(2)}$ are assigned to the remaining positions (again in the randomized order).

The inclusions in p_l that mark a non-empty codeblock contribution, i.e., *FI* and *PI*, are assigned the length and number of new coding passes of the non-empty codeblock contributions of the correct inclusion vector v in the order in which these contributions appear in v (these CCP lengths and number of coding passes may be subject to transformation later, or maybe have already been transformed). All first inclusions (*FI*) in $\hat{v}^{(1)}$ are assigned the number of leading zero-bitplanes in the order of *FI*-inclusions in v. After all packets have been processed, the new header information is written. If the key that has been used for the permutation is known, this procedure is reversible. Note that the permutation in this approach crosses subband boundaries: the inclusion information is reassigned over all codeblocks in the packet.

For empty packets, no permutation is possible. For these packets, the inclusion information only needs to be updated based on the inclusion information in the previous packet, according to the possible transitions.

To illustrate the process of format-compliant permutation we give an example. Let $v_{p_l} = [FI, NI, NI]$ be the inclusion vector of the first candidate package p_l in a resolution with three codeblocks c_0, c_1, c_2. After permutation the new inclusion information is $\hat{v}_{p_l} = [NI, NI, FI]$. The length of the codeblock contribution and number of leading zero-bitplanes is transferred from c_0 to c_2. In the next packet p_{l+1} let the real inclusion information be given by $v_{p_{l+1}} = [PI, FI, NI]$. This vector is split into $v^{(1)}_{p_{l+1}} = [FI, NI]$ and $v^{(2)}_{p_{l+1}} = [PI]$. Considering the "faked" inclusion information $\hat{v}_{p_l}$ of the previous packet, the positions of codeblocks c_0 and c_1 are the candidate positions for inclusions of type *FI* and *NI*. $v^{(1)}_{p_{l+1}}$ is permuted to form $\hat{v}^{(1)}_{p_{l+1}} = [NI, FI]$ and the new inclusion information is assigned to the respective positions of c_0 and c_1. The length of the contribution and the number of coding passes and leading zero-bitplanes are updated for the non-empty contribution. In this example, $v^{(2)}_{p_{l+1}}$ only has one element which has to be assigned to the position of codeblock c_2 to form a consistent sequence of inclusion information. The length of the contribution and the number of coding passes is updated for this codeblock. The new inclusion information for p_{l+1} is $[NI, FI, PI]$.

We now turn to the investigation of the number of possible permutations. For the lower resolutions, there will typically be only few codeblocks that contribute to each packet, so the number of permutations will be very limited. As the permutation of packet headers is only used in conjunction with the encryption of packet bodies, this is not a problem. The fingerprint that could be derived from the inclusion information in the lower resolutions is not very distinctive, the main point is to destroy fingerprints in the higher resolutions.

We can give the number of possible permutations for a packet p_l which contains the codeblock contributions for layer l (for a specific resolution). Let $|C_{p_l}|$ be the total number of codeblocks that are relevant for p_l, and $|FI_{p_l}|$, $|NI_{p_l}|$, $|PI_{p_l}|$, $|PN_{p_l}|$ the number of codeblocks in p_l with inclusion type *FI*, *NI*, *PI* and *PN*, respectively. If p is the first packet to be permuted, we have no restrictions in the possible permutations. Furthermore, in this case the inclusion information in p will consist of either only *FI* and *NI* or of *PN* and *PI* (because otherwise a previous packet would have been the first to be permuted).

Cblk. size	rate (bpp)	layers	perm.
64×64	3	32	10^{217}
32×32	3	32	10^{938}
64×64	3	12	10^{56}
32×32	3	12	10^{257}
64×64	0.25	32	10^{42}
32×32	0.25	32	10^{146}

Table 1: Number of possible format-compliant permutations of the inclusion information for Lena (*wlev*= 5)

Without loss of generality, we assign the positions for inclusions of type *FI* and *PI*. The number of possible permutations is then given as:

$$\binom{|C_{p_l}|}{|FI_{p_l}|}\binom{|C_{p_l}|}{|PI_{p_l}|}. \tag{1}$$

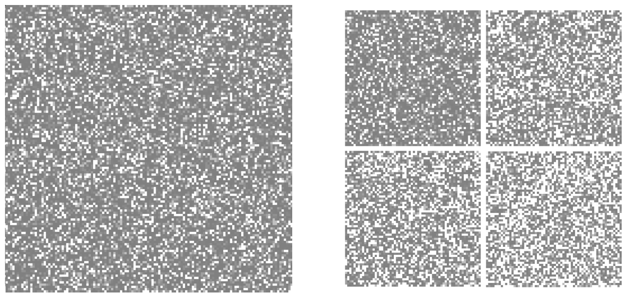
If p is a packet that pertains to a higher layer than the first candidate, the inclusion information of the preceding packet p_{l-1} has to be taken into account. Inclusions of type *FI* and *NI* in p can go into positions that are included as *NI* in p_{l-1}. The rest of the positions can be assigned to inclusions of type *PI* and *PN* in p_l. The number of possible permutations is determined by

$$\binom{|NI_{p_{l-1}}|}{|FI_{p_l}|}\binom{|FI_{p_{l-1}}| + |PI_{p_{l-1}}| + |PN_{p_{l-1}}|}{|PI_{p_l}|}. \tag{2}$$

The actual number of permutations for a given input image depends on a variety of factors. The used compression parameters influence the number of codeblocks that are available for permutation in the first place. With small codeblock sizes the number of available codeblocks increases. If the number of quality layers is increased, then there are also more packets and therefore more permutations can be performed. The bitrate with which the image is encoded also influences the number of packets that are included in the codestream. Finally, the number of permutations that can be applied to the packets of a resolution is influenced by the point at which the first candidate packet is found, and by how diverse the inclusion information is in this packet and the following packets. If all the codeblocks of a resolution always have the same inclusion information in each packet, then no permutation of inclusion information is possible. Luckily, this case is extremely unlikely in practice. Table 1 shows the number of possible permutations (perm.) for the Lena image (512×512 pixels) with different compression settings (codeblock size, rate, number of quality layers).

3.4 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 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 order in which they are applied is arbitrary, only decoding has to apply the reverse transformations in the reverse order.



(a) Number of LZB transformed, $wlev = 0$

(b) Number of LZB transformed, $wlev = 1$

Figure 2: Visualization of attack after LZB transformations (for wlev=0 and codeblock size 4×4)

4. EVALUATION

Generally it should be noted that the proposed key-dependent transformations use permutations, which are principally vulnerable to known-plaintext attacks. Therefore, the key for the transformation of header information should under no circumstances be derived from the key that is used for the encryption of the packet bodies.

In the following, we evaluate the proposed scheme in the two – very different – scenarios. First, we look at format-compliant packet body encryption schemes that aim at providing full confidentiality. In these scenarios we deal with a situation in which all packet body data are encrypted. The motivation to use header protection in this scenario is to avoid the creation of a fingerprint.

The other scenario is format-compliant partial / selective encryption that aims at a reduction in computational complexity. As portions of the data remain in plaintext, partial / selective encryption schemes are not well suited for full confidentiality (even if the data cannot be used to create a visual reconstruction, they can always be used as a fingerprint). The aim here is to introduce sufficient reduction of the visual quality to prevent a pleasant viewing experience. Traditional approaches encrypt packet bodies in a format-compliant way. We investigate if a combination with header protection can improve such a scheme.

4.1 Confinement of Information Leakage

The visual information contained in the packet header as the number of leading zero bitplanes is a severe security problem for applications that require full confidentiality, as this information is preserved in the encrypted bitstream after packet body encryption. The proposed header protection scheme solves this security problem: the LZB information is completely destroyed by the transformation. Figure 2 shows the same LZB-images for Lena as Figure 1, but with the headers transformed. As can be seen, no visual content is discernible anymore.

As discussed in Section 2, the information contained in the header information can also be used as a distinctive fingerprint. This fingerprint remains even if all of the packet body data is encrypted. Figure 3 illustrates this point. We have tested 175 images, all taken with the same camera model and cropped to 512×512 pixels at 8 bpp. All of the images are encoded with JPEG2000 at a bitrate of 0.25 bpp with 32

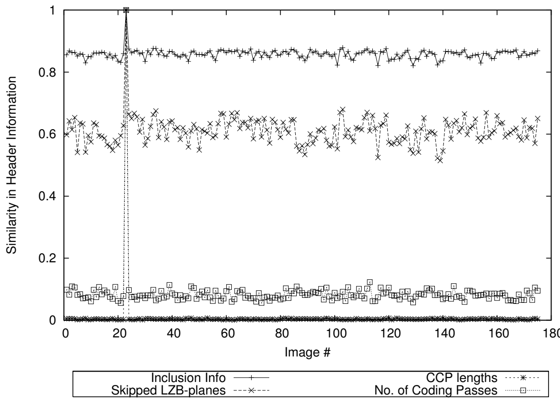


Figure 3: Comparison of similarity in header information for 175 images

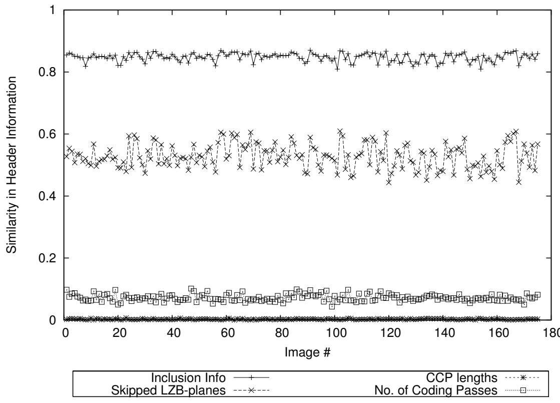


Figure 4: Comparison of similarity in header information for transformed header information

quality layers and a codeblock size of 32×32. The header information in all the packets of each image is recorded. We then compare the header information of a single image from the set to the header information of each other image in the set. The ratio of the number of items in the header information that have the same value (at the same position) to the total number of items is recorded. In the plots, the ordinate shows this value for each class of header information and each image. Note that for CCP lengths and number of coding passes we ignore positions in the header for which the information is 0 for both the reference and comparison image. Figure 3 shows the similarity in header information of one image (# 23) with the other images. It can be seen that the similarity measures to other images are confined within a certain range of variance. It is not surprising to see that the similarity of the CCP-lengths is very small for differing images. Interestingly, the number of corresponding items in the inclusion information is very large. This is due to the fact that for the inclusion information we also counted all the inclusions of type *NI* which at this bitrate occur a lot. The variance of the similarity of the inclusion information is confined relatively strictly and therefore also the inclusion information may serve as a discriminating feature.

Obviously the situation will be different if the reference image is re-encoded with different compression parameter settings. But as this illustration shows, any class of header information can be used to link a known JPEG2000 plaintext to a packet-body encrypted ciphertext. For many applications that require full confidentiality such a leak of information constitutes a major compromise of security.

The proposed transformations can be used to prevent the creation of a fingerprint which uses the details of the packet headers, such as proposed by [10, 3, 17]. Figure 4 shows the same comparison as Figure 3, but this time with the header information of the reference image transformed. It can be observed that the proposed transformation methods obstruct the identification of the image in the set of 175 images: the transformed header bears no similarity to the original header. Only for the codeblock lengths a minute trace remains. This is due to packets of the lowest two resolutions which only contain a single codeblock each. For the used compression settings no transformation was possible for these packets.

Note that as the proposed scheme preserves packet boundaries and does not merge or split the data in the packets, some information does of course remain. A fingerprint that uses the size of the individual packets can still be created. Furthermore, the number of inclusions of each inclusion type stays the same as in the plaintext image. This information could be used to obtain a fingerprint, albeit a much weaker one than if the order of inclusions was known. If packet boundaries were crossed and furthermore inclusion information was split up among codeblocks, a possible fingerprint would be further weakened. However, the downside would be a loss in semantics for the encrypted version (which would make rescaling more unreliable, for example). The highly discriminative fingerprints based on the detailed packet header information are successfully prevented.

4.2 Improvement of Partial / Selective Packet Body Encryption

In this section we evaluate if header transformation can improve partial / selective encryption schemes. Before we go into detail, we need to look at the computational demands of the header transformations compared to packet body encryption. As the array-based permutations are all of low complexity, the header transformations are computationally cheap. The necessary operations in each substep are the generation of a random number and exchanging the places of two items in the array. The cipher used for packet body encryption will usually be computationally more demanding. Furthermore, for practical compression settings the ratio of packet header data to packet body data is very small, as illustrated by Table 2 for the Lena image, which shows the percentage of packet body data and packet header data, respectively, to the size of the total bitstream (compression at full rate, nearly lossless).

Typical partial / selective encryption schemes for JPEG2000, like the one proposed by [13], encrypt a certain percentage of the packet body data, starting from the beginning of the bitstream. This can be done either in layer or resolution progression mode (for a comparison see [13]). Even if small portions of the bitstream are encrypted, high distortion in visual quality can be achieved.

A straightforward attack is discussed by [13]: The concealment attack discards all encrypted parts of the bitstream.

Cblk. Size	Layers	Packet Header	Packet Body
64×64	16	0.84%	99.02%
64×64	32	1.15%	98.71%
32×32	32	2.97%	96.98 %
16×16	32	7.94%	91.94%

Table 2: Distribution of data between packet header and packet body in percent of the total bitstream

The authors simulate the attack by using JPEG2000 error concealment. During encoding, an error resilience segmentation symbol is inserted at the end of each bitplane. If the decoder cannot decode the segmentation symbol, an error has been detected. We have adapted the error concealment to handle encrypted bitstreams: If the segmentation symbol cannot be decoded correctly, then the affected codeblock contribution is discarded as a whole ("zero concealment"). The attack lowers the level of reduction in visual quality for the encryption scheme. This is illustrated by Figure 5, for which 1% of the packet body data has been encrypted: (a) shows the direct reconstruction without error concealment. It can be seen that there is a high degree of visual distortion. However, as (b) shows, the zero concealment achieves a reconstruction that is of a higher visual quality.

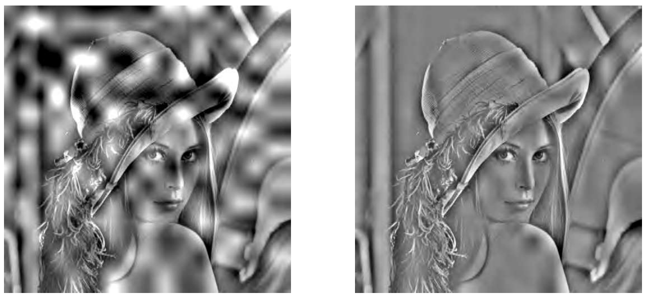

(a) direct reconstruction (b) zero concealment

Figure 5: 1% of body data encrypted

The concealment attack can be completely prevented if, apart from a part of the packet body data, all packet headers are encrypted. The header transformation of a packet causes the error concealment procedure to regard the packet body data as erroneous and discard it. The result is shown for the Lena image at a compression rate of 0.25 bpp in Figure 6. As the peak-signal-to-noise-ratio (PSNR) is not suitable as a measure for images of such low quality, we have used the edge similarity score (ESS) proposed by [12]. The ESS is computed by block-based gradient comparison and ranges, with increasing similarity, between 0 and 1. We use the weights and blocksizes proposed by [12] in combination with Sobel edge detection. Figure 6 shows that for low encryption rates the partial packet body approach preserves a lot of edge information. If the packet body approach is combined with full packet header protection, then the edge information disappears from the reconstruction even for very small packet body encryption rates, and a high minimum level of distortion can be guaranteed. Some visual examples are shown in Figure 7. It can be seen that compared

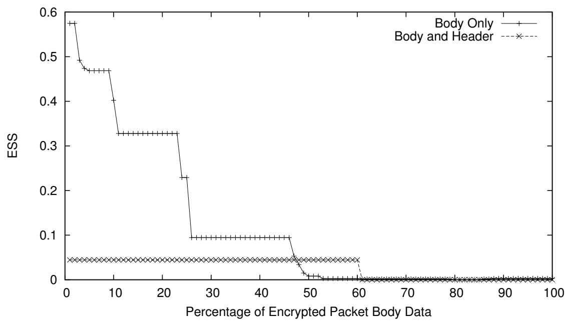
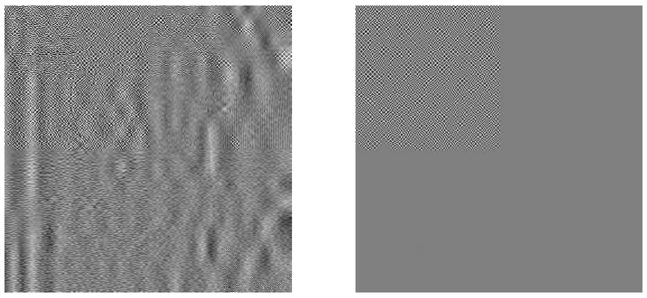


Figure 6: ESS comparison of concealment attack with encrypted header and without encrypted header



(a) direct reconstruction

(b) zero concealment

Figure 7: 1% of body data encrypted plus all packet headers transformed

to the packet body only approach, the additional packet header encryption prevents both, direct reconstruction and zero concealment attacks.

Of course, this comes at the cost of encrypting all headers. But, as we discussed above, for practical coding settings, the size of the packet header data will be only a small fraction of the packet body data. So, rather than encrypting a larger portion of the packet body data, it is advisable for most compression settings to encrypt all packet headers, as for this approach the costs are low and the distortion is high. As an example we take the 512×512 pixel Lena image at 0.25 bpp, 32 quality layers and a codeblock size of 64×64 pixels. In this case, the packet header data is 3.5% and the body data is 94.7% of the total bitstream (the rest is main header data). As Figure 6 illustrates, the distortion introduced by 1% packet body encryption and full packet header encryption is comparable to the distortion introduced by approximately 45% packet body encryption. Encrypting 45% of the packet body data in these settings amounts to encrypting 42% of the total bitstream. The combined approach of encrypting 1% of the body data and transforming all header data amounts to dealing with data that corresponds to 4.4% of the total bitstream.

In combination with the encryption of a small portion of the packet body data, the packet header protection scheme can be used in applications that do not strive for full confidentiality. The overall number of possible transformations is too high for a successful brute-force attack. Only the packets in the lowest resolutions have few enough codeblocks to make trying all possible headers an option. So while an attacker could gain an idea of the visual content by attacking the packets of the lower resolutions for which only the packet headers but not the packet bodies have been protected, the full visual quality version of the original content remains protected. This is also true in the presence of a preview image that could have been obtained from side information. As [15] points out, many partial encryption schemes are vulnerable to low complexity attacks based on the availability of such side information. In the context of the proposed scheme, a low quality preview image cannot be used to reconstruct the JPEG2000 packet header information. We have also found no way to reconstruct the packet header from the unencrypted packet body data.

The question arises whether, as the header transformation guarantees a high level of reduction in visual quality, it would be a good approach to only use packet header protection for the whole bitstream and leave the packet bodies in plaintext. Such an approach would only be a feasible option if the desired reduction in visual quality was very low (i.e., if the desired encryption scheme is what is commonly called "transparent"[11]). This is also because the lower resolutions in the bitstream only have a small number of possible transformations. For practical coding settings, the lowest resolution will often only be constituted of a single codeblock. In this case, the packets of this resolution will invariably only have a single entry of inclusion information, CCP length, LZB information and number of truncation points. Only the higher resolutions offer a large number of possible transformations. For high visual distortion it is therefore inevitable to use packet body based encryption at the beginning of the bitstream.

5. CONCLUSION

We have proposed a fully format-compliant protection scheme for JPEG2000 packet headers. In the context of encryption schemes that aim at sufficient distortion of visual quality rather than at providing full confidentiality, we have shown that the proposed scheme can help to improve the level of guaranteed visual distortion.

More importantly, the proposed scheme can be used to confine the information leakage that is present in all encryption scheme that selectively encrypt packet body data. The visual information contained in the JPEG2000 packet header is destroyed. Furthermore, the strong fingerprint based on the details of the packet headers is prevented by the scheme. As the scheme preserves packet boundaries, fingerprints based on more general information like overall packet lengths are not destroyed. However, if format-compliance is desired in a sense that allows to perform tasks like rate adaption in the encrypted domain, the information needed for these tasks will always need to be preserved to some extent. Therefore, while the proposed scheme significantly improves the security of format-compliant encryption schemes that rely on the encryption of packet body data, the combined schemes can never be as secure as full encryption. The straightforward approach to encrypt the *full* bitstream with a cryptographically strong cipher is and remains the *only* option that is secure in the strong cryptographic sense.

6. REFERENCES

[1] V. Conan, Y. Sadourny, K. Jean-Marie, C. Chan, S. Wee, and J. Apostolopoulos. Study and validation of tools interoperability in JPSEC. In A. G. Tescher, editor, *Applications of Digital Image Processing XXVIII*, volume 5909, page 59090H. SPIE, 2005.

[2] R. H. Deng, W. S. Di Ma, and Y. Wu. Scalable trusted online dissemination of JPEG2000 images. *Multimedia Systems*, 11(1):60 – 67, Nov. 2005.

[3] 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 B. Günsel, A. K. Jain, A. M. Tekalp, and B. Sankur, editors, *Proc. Multimedia Content Representation, Classification and Security, MRCS 2006*, volume 4105 of *Lecture Notes in Computer Science*, pages 282–289, Berlin, Heidelberg, New York, Tokyo, Sept. 2006. Springer-Verlag.

[4] F. Dufaux and T. Ebrahimi. Securing JPEG2000 compressed images. In A. G. Tescher, editor, *Applications of Digital Image Processing XXVI*, volume 5203, pages 397–406. SPIE, 2003.

[5] R. Grosbois, P. Gerbelot, and T. Ebrahimi. Authentication and access control in the JPEG2000 compressed domain. In A. Tescher, editor, *Applications of Digital Image Processing XXIV*, volume 4472 of *Proceedings of SPIE*, pages 95–104, San Diego, CA, USA, July 2001.

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

[7] ISO/IEC 15444-4. Information technology – JPEG2000 image coding system, Part 4: Conformance testing, Dec. 2004.

[8] ISO/IEC 15444-8. Information technology – JPEG2000 image coding system, Part 8: Secure JPEG2000, Apr. 2007.

[9] 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)*, volume III, pages 205–208, Barcelona, Spain, Sept. 2003.

[10] 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*, pages 46–49, New York, NY, USA, 2001. ACM Press.

[11] B. M. Macq and J.-J. Quisquater. Cryptology for digital TV broadcasting. *Proceedings of the IEEE*, 83(6):944–957, June 1995.

[12] Y. Mao and M. Wu. Security evaluation for communication-friendly encryption of multimedia. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'04)*, Singapore, Oct. 2004. IEEE Signal Processing Society.

[13] R. Norcen and A. Uhl. Selective encryption of the JPEG2000 bitstream. In A. Lioy and D. Mazzocchi, editors, *Communications and Multimedia Security. Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security, CMS '03*, volume 2828 of *Lecture Notes on Computer Science*, pages 194 – 204, Turin, Italy, Oct. 2003. Springer-Verlag.

[14] Y. Sadourny and V. Conan. A proposal for supporting selective encryption in JPSEC. *IEEE Transactions on Consumer Electronics*, 49(4):846– 849, Nov. 2003.

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

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

[17] 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*, pages 329–337. IEEE Computer Society Press, Mar. 2005.

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

[19] S. Wee and J. Apostolopoulos. Secure scalable streaming enabling transcoding without decryption. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'01)*, Thessaloniki, Greece, Oct. 2001.

[20] S. Wee and J. Apostolopoulos. Secure scalable video streaming for wireless networks. In *Proceedings of the 2001 International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001)*, Salt Lake City, Utah, USA, Apr. 2001. invited paper.

[21] H. Wu and D. Ma. Efficient and secure encryption schemes for JPEG2000. In *Proceedings of the 2004 International Conference on Acoustics, Speech and Signal Processing (ICASSP 2004)*, pages 869–872, May 2004.

[22] M. Wu and V. Mao. Communication-friendly encryption of multimedia. In *Proceedings of the IEEE Multimedia Signal Processing Workshop, MMSP '02*, St. Thomas, Virgin Islands, USA, Dec. 2002.

[23] Y. Wu and R. H. Deng. Progressive protection of JPEG2000 codestreams. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'04)*, Singapore, Oct. 2004. IEEE Signal Processing Society.