Region of Interest Signalling for Encrypted JPEG Images

Dominik Engel
Salzburg University of Applied Sciences
Urstein Süd 1
Puch/Salzburg, Austria
dengel@en-trust.at

Andreas Uhl
University of Salzburg
Dept. of Computer Sciences
Jakob-Haringer-Str. 2
Salzburg, Austria
uhl@cosy.sbg.ac.at

Andreas Unterweger
University of Salzburg
Dept. of Computer Sciences
Jakob-Haringer-Str. 2
Salzburg, Austria
aunterweg@cosy.sbg.ac.at

ABSTRACT

We propose and evaluate different methods to signal position and size of encrypted RoIs (Regions of Interest) in JPEG images. After discussing various design choices regarding the encoding of RoI coordinates with a minimal amount of bits, we discuss both, existing and newly proposed approaches to signal the encoded coordinates inside JPEG images. By evaluating the different signalling methods on various data sets, we show that several of our proposed encoding methods outperform JBIG in this special use case. Furthermore, we show that one of our proposed signalling methods allows length-preserving lossless signalling, i.e., storing RoI coordinates in a format-compliant way inside the JPEG images without quality loss or change of file size.

Categories and Subject Descriptors

E.2 [**Data**]: Data Storage Representations—*Object representation*; E.4 [**Data**]: Coding and Information Theory—*Data compaction and compression*; I.4.2 [**Image Processing and Computer Vision**]: Compression (Coding)—*JPEG*

General Terms

Algorithms, Theory, Measurement

Keywords

JPEG, Region of Interest, Coordinates, Encoding, Signalling

1. INTRODUCTION

In the last decade, a large number of region of interest encryption approaches have been proposed, especially for image and video formats using DCT-domain-based compression, like JPEG [12]. Although the human eye is capable of detecting encrypted picture regions easily, state-of-the-art software is not. There have been attempts to detect encrypted picture regions automatically [3], i.e., without the need to signal them explicitly. However, despite their reported near-100% accuracy, it is clear that perfect detection is not possible, albeit necessary for correct decryption in most cases.

Therefore, there is an immanent need to store the encrypted RoIs' coordinates inside the JPEG file in order to have them available during decryption. As the JPEG file format has no means of signalling encrypted regions, unlike JPEG2000 [1], different methods of encoding these coordinates have to be evaluated and a detailled analysis of possible signalling methods is required.

All RoI encryption approaches for JPEG proposed so far handle RoI signalling in one of three ways. The first method involves using JPEG comment segments with an unspecified coordinate encoding [2], which is straight-forward, but does not take into account that some applications do not tolerate size changes of the JPEG file. Thus, we explore different options and present solutions for a variety of typical practical contraints.

The second and most common signalling method relies on an external signalling channel [4]. As signalling RoIs in the JPEG image itself has significant advantages as compared to using a separate channel, this paper proposes and evaluates possibilities to store coordinates of encrypted RoIs inside the JPEG images themselves.

Finally, the third signalling method is to omit signalling details altogether [13], which can make decryption impossible or dependent on human RoI identification. As this is not acceptable in most cases, an analysis of encoding and signalling methods for encrypted RoIs in JPEG images is required.

This paper is structured as follows: In Section 2, we discuss design choices for RoI coordinate encodings and select a subset thereof for further evaluation. In Section 3, we propose different methods to signal the encoded coordinates inside a JPEG file. Subsequently, in Section 4, we evaluate all encoding and signalling methods in order to find appropriate combinations for different use cases before we conclude the paper in Section 5.

2. ROI COORDINATE ENCODING

Before the RoI coordinates can be signalled, they have to be encoded appropriately. As the number of signalled bits may be limited or even influence the picture quality, depending on the signalling method used, a compact encoding is desired. In this Section, we discuss several design choices for encodings, aiming at listing a set of practically useable encodings to be evaluated in Section 4.

As most state-of-the-art encryption approaches for JPEG

IH&MMSec'13, June 17–19, 2013, Montpellier, France.
Copyright 2013 ACM 978-1-4503-2081-8/13/06 ...$15.00.

Name	**Explicit**	**Value encoding**	**Differential**	**Bits per RoI**	**Overhead bits per file**
Bitmap	–	Fixed length	N/A	0	n_{iMCU}
List	✓	Fixed length	–	$2 \cdot \lceil log_2(n_{iMCU}+1) \rceil$	$2 \cdot \lceil log_2(n_{iMCU}+1) \rceil$
VList	✓	Exp. Golomb	–	Variable	2
DList	✓	Signed Exp. Golomb	✓	Variable	2
ACBitmap	–	Fixed + ABAC	N/A	0	Variable
ACList	✓	Fixed + ABAC	–	Variable	Variable
ACVList	✓	Exp. G. + ABAC	–	Variable	Variable
ACDList	✓	S. Exp. G. + ABAC	✓	Variable	Variable
JBitmap	–	JBIG	–	Variable	12 (header only)

Table 1: List of coordinate encodings to be evaluated and their respective storage requirements

operate on a block [15] or iMCU (interleaved Minimum Coding Unit, multiple luminance and the corresponding chrominance blocks) level [22], coordinates are limited to iMCU granularity. Note that this limitation is also imposed – often self-imposed [3] – on format-independent encryption approaches which operate in the image domain.

Furthermore, chrominance subsampling is assumed to be 4:2:0 [14] as it is the default setting in the JPEG reference software and widely used [20]. This enforces a fixed iMCU size of six blocks, four of which are luminance blocks [12], limiting the coordinate granularity to rectangular image blocks of $16 \cdot 16$ pixels size.

Subsequently, the following variables are used: w and h denote a picture's width and height in pixels, respectively. Furthermore, the width and height in iMCUs are defined as $w_{iMCU} = \lceil \frac{w}{16} \rceil$ and $h_{iMCU} = \lceil \frac{h}{16} \rceil$, respectively. In addition, $n_{iMCU} = w_{iMCU} \cdot h_{iMCU}$ denotes the total number of iMCUs in a picture. Finally, n_{RoI} specifies the number of RoIs to be encoded. All coordinate encodings described in the subsequent subsections are summarized in Table 1 using the aforementioned variables.

2.1 Implicit vs. explicit encoding

Coordinates can be encoded either implicitly or explicitly. While implicit encoding entails deriving the actual coordinates locally, e.g., from the position of bit patterns in a bitmap, explicit encoding stores the actual coordinates globally so that they can be read directly. Hence, the simplest form of implicit encoding, i.e., a bitmap for all iMCUs where a zero bit means "not encrypted" and a one bit means "encrypted", requires n_{iMCU} bits to be stored (see Table 1: "Bitmap").

In contrast, explicit coordinate encoding requires storing a list of coordinates, specifying the location and size of each RoI. Both, location and size, are described by a horizontal and vertical component, referred to as X and Y coordinate, respectively, yielding four coordinates in total.

In addition, it is necessary to specify a special coordinate signalling the end of the coordinate list. For the sake of simplicity and practicality, we subsequently use a RoI with a size of zero to signal the end of the list. This is reflected in the per-file overhead of all explicit encodings listed in Table 1 accounting for the additional end-of-list entry as storing n_{RoI} RoIs requires $n_{RoI} + 1$ list entries in total. Each list entry consists of 4 coordinates, 2 of which are X and Y coordinates, respectively.

2.2 Component vs. index encoding

Although separate X and Y coordinates allow locating the encrypted RoIs easily, two components (X and Y) need to be stored to specify one location. When using a fixed bit length per component, the X and Y coordinate require $\lceil log_2(w_{iMCU} + 1) \rceil$ and $\lceil log_2(h_{iMCU} + 1) \rceil$ bits of space, respectively.

Alternatively, an index can be assigned to each iMCU, starting with zero for the top-left-most iMCU and increasing in the left-to-right and top-to-bottom direction. This way, a location identified by two components (X and Y) can be specified by a single index which requires $\lceil log_2(n_{iMCU} + 1) \rceil$ bits when using a fixed bit length per index. Note that this is always shorter than or in the worst case as long as signalling two separate components since $\lceil log_2(n_{iMCU} + 1) \rceil = \lceil log_2(w_{iMCU} \cdot h_{iMCU} + 1) \rceil \leq \lceil log_2(w_{iMCU}) + log_2(h_{iMCU}) + 1 \rceil \leq \lceil log_2(w_{iMCU} + 1) \rceil + \lceil log_2(h_{iMCU} + 1) \rceil$, which is the number of bits required for two separately stored X and Y coordinates. Thus, index encoding is to be preferred over component encoding and all explicit encodings listed in Table 1 encode iMCU indices instead of X and Y coordinates.

2.3 Fixed-length vs. variable-length encoding

As the picture width and height are known, the maximum number of bits required to encode one iMCU index can be determined easily. If this fixed bit length is used for all indices, encoding one RoI requires $2 \cdot \lceil log_2(n_{iMCU}) \rceil$ bits in total (see Section 2.2), the factor of two being required to account for both, the location and size of the RoI (see Section 2.1). This way, each encoded RoI requires the same number of bits, regardless of its own size and location (see Table 1: "List").

As RoIs usually do not span the whole picture, using a constant number of bits which allows specifying the whole picture size can be disadvantageous. Similarly, RoI locations on the top-left require a high number of bits, although their corresponding iMCU start indices are small. Hence, the use of variable-length encoding for both iMCU indices, specifying the encrypted RoI's location and size, is to be evaluated. One method for variable-length coding are Exponential-Golomb codes as used e.g., for encoding a subset of H.264 syntax element values [19]. As a RoI's position and size (represented as iMCU indices) are always positive, a zeroth order (i.e., $k = 0$) unsigned Exponential-Golomb code ("ue(v)" following the notation of the H.264 standard [11]) can be used to encode them. Table 2 shows examples of values and their respective encoded bit representation.

As can be seen, a value of zero can be signalled using one bit. Hence, an end-of-list entry (with position and size being zero) can be signalled using two bits (see Table 1). Gener-

Value	ue(v) code word	se(v) code word
...	–	...
-4	–	0001001
-3	–	00111
-2	–	00101
-1	–	011
0	1	1
1	010	010
2	011	00100
3	00100	00110
4	00101	0001000
...	...	...

Table 2: List of exemplary values and their respective zeroth order Exponential-Golomb code words. Hyphens denote invalid value ranges

ally, any positive integer value x requires $2 \cdot \lceil log_2(x+2)\rceil - 1$ bits. Thus, one iMCU index requires a maximum of $2 \cdot \lceil log_2(n_{iMCU}+2)\rceil - 1$ bits. As the actual number of bits can be smaller, depending on the actual iMCU indices to be encoded, the storage requirements per RoI are variable when using Exponential-Golomb encoded list entries (see Table 1: "VList" for variable-length coded list), possibly reducing the number of stored bits compared to fixed-length encoding.

2.4 Differential encoding

Although variable-length coding reduces the storage requirements when encoding small indices, the converse is true for large indices, i.e., indices identifying iMCUs at the bottom-right of a picture. In order to overcome this drawback, each index can be stored relative to its predecessor, replacing the actual value to be encoded by a differential value which is very likely to be smaller. For example, a location/size pair (l_2, s_2) can be encoded as $(l_2 - l_1, s_2 - s_1)$ relative to its preceding location/size pair (l_1, s_1). As all RoIs are known, their order in the RoI list can be chosen so that the differential values to be encoded are minimal in terms of size.

However, it is not guaranteed that there is an order of entries in the RoI list so that all differences are positive, thus requiring the ability to encode negative differences as well. Signed Exponential-Golomb codes which support both, positive and negative values, are described in the H.264 standard [11]. Following the latter's notation, such zeroth order codes are referred to as "se(v)". Table 2 shows examples of values and their respective encoded bit representation.

In general, any integer value x requires $2 \cdot \lceil log_2(2 \cdot |x| + 2)\rceil - 1$ bits as signed Exponential-Golomb code word, which is more than the amount required for the respective unsigned Exponential-Golomb code word. Nonetheless, we include this encoding approach as its storage requirements depend on the RoI's coordinates' differences (see Table 1: "DList" for differentially encoded list) which depend on the values and ordering of the RoIs, unlike all other encodings.

2.5 Entropy coding

Each of the encodings described above makes use of different representations and/or properties of the list of RoI coordinates. However, none of them aims at effectively eliminating redundancy. Thus, a modified version of each encoding is included in Table 1 which essentially adds an entropy coding step after the original encoding process, indicated by a "C" (for compressed) prefix in the encoding's name.

Arithmetic coding [24] (prefixed with an additional "A") is chosen for the entropy coding step as it theoretically allows for quasi optimal, i.e., close-to-entropy, performance. As the number of different values to be encoded is equal to n_{RoI} for n_{ROI} RoI location/size pairs and smaller than or equal to $2 \cdot n_{RoI}$ for separately encoded location and size values, binary arithmetic coding (abbreviated BAC in Table 1) calculated in fixed-precision integer arithmetic as described in the JPEG standard [12] is evaluated.

As signalling the symbols' probabilities (or the corresponding subintervals) would require additional bits, adaptive coding, i.e., the dynamic adjustment of the symbol probabilities, is used to optimize coding efficiency [19]. Starting with equal probabilities for both symbols, zero and one, the subinterval ranges are adjusted according to the changing symbol frequencies during encoding. Note that end-of-stream markers can be omitted as the decoding process can stop the arithmetic decoding process as soon as the end-of-list marker (a RoI with location and size zero) is found.

2.6 Bi-level image compression

As the implicitly encoded bitmap described in Section 2.1 is in fact a bi-level image, the use of a compressor which is optimized for this type of images has to be evaluated for comparison. Due to its widespread use, we choose the JBIG compression standard [9] in combination with one of its application profiles [10] for this task (see Table 1: "JBitmap" for JBIG-compressed bitmap).

In order to compensate for its relatively large file header with a total size of 20 bytes, we shorten the former by the eight bytes which signal the image's width and height as they can also be derived otherwise, e.g., from the JPEG picture. This reduces the total per-file overhead to twelve bytes, thus allowing for a fairer comparison.

2.7 Summary

A number of choices have to be made when designing an encoding for a list of RoIs, few of which are clear without prior evaluation. As outlined in Section 2, encoding iMCU indices always requires less than or as many bits as encoding separate X and Y coordinates, Thus, all encodings to be evaluated encode iMCU indices. As most other design criteria of possible encodings depend on either the number and/or size of the RoIs and/or the picture, a selected subset of possible encodings (see Table 1) covering all of the aforementioned criteria has to be evaluated in Section 4.

3. ROI SIGNALLING

The encoded RoIs' coordinates need to be signalled in some form in order to identify the RoIs at a later point in time, e.g., during the decryption process. Thus, in this Section, we propose a number of different ways to store the encoded RoI coordinates directly inside the JPEG file. In order to account for the different needs of conceivable use cases, the proposed signalling methods are chosen to cover a number of different combinations of the following aspects:

1. **Format compliance**: The strict fulfillment of all syntactical and semantical requirements imposed by the JPEG standard [12]

2. **Losslessness**: The exact preservation of all (visible) picture data

3. **Availability**: The guarantee that the proposed method will work on every JPEG picture

4. **Length-preservation**: The guarantee that the picture's file size does not change (suitable for length-preserving encryption methods like [22])

Furthermore, the capacity, i.e., the amount of storable bits, of each signalling method is given. Note the capacity of some of the proposed methods depends on the picture and/or its metadata. As the number of RoIs is usually not known in advance for all pictures, all methods need to be evaluated in terms of usability for storing encoded RoI coordinates as proposed in Section 2, which is done in Section 4.

All proposed methods are described with regards to the aforementioned aspects and summarized in Table 3 for convenience. For reasons of practicality, we assume that all JPEG pictures are Baseline JPEG pictures [12] with three color components – Y, Cb and Cr, i.e., one luminance and two chrominance components. Note that most methods will, however, work with differently coded JPEG pictures (e.g., arithmetically coded ones) as well.

3.1 Use of COM and APP segments

The first method, the insertion of a COM (Comment) segment into the JPEG file according to Annex B of the JPEG standard [12], has already been proposed by others (e.g., [2]). One COM segment may contain up to 65533 payload bytes, plus its marker (2 bytes) and length field (2 bytes), totalling to 65537 stored bytes. As the number of COM segments is theoretically unlimited, so is the total capacity of this signalling method. Signalling n bits requires $n_{COM} = \left\lceil \frac{\lceil \frac{n}{8} \rceil}{65533} \right\rceil$ COM segments with a total of $(n_{COM} - 1 + \epsilon) \cdot 65537 + 4 + \lceil \lceil \frac{n}{8} \rceil \ mod \, 65533 \rceil$ bytes, where mod denotes the integer modulus operator and ϵ is a correction factor of 1, if there is no remainder (of the modulus operation), and 0 otherwise. The rounding to full bytes is due to the fact that a COM segment's length field is expressed in bytes, not bits.

As an alternative to the COM segment, an Application Data (APP) segment can be used, which is equivalent in terms of structure. As there are 16 different APP markers, it is theoretically possible to encode four more bits into an APP segment than into a comment segment of equal total size. As the capacity is otherwise the same, signalling n bits in APP markers requires $n_{APP} = \left\lceil \frac{\lceil \frac{n}{4} \rceil}{2 \cdot 65533.5} \right\rceil$ APP segments with a total of $(n_{APP} - 1 + \epsilon) \cdot 65537 + 4 + \left\lceil \frac{\lceil \lceil \frac{n}{4} \rceil \ mod \, (2 \cdot 65533.5) \rceil}{2} \right\rceil$ bytes. Due to the additional 4 bits per segment, APP segment signalling is to be preferred over COM segment signalling in terms of capacity. However, there may already be APP segments in the JPEG file, in which case the gain in capacity may be reduced. Moreover, there may be a border case in which all different APP segment types are already present in the file, making it impossible to store any data in this way.

One commonly used APP segment type is APP$_1$, typically storing data in the Exchangeable Image File Format (EXIF) [6]. If such data is present, but not crucial for further processing, it can be replaced by encoded RoI coordinates. However, this method of stripping EXIF data depends on the presence of the latter and is usally very limited in terms of capacity. A more detailed description of this method and its capacity is provided in [5], which is why it is not evaluated separately herein.

3.2 Use of dummy tables

Although JPEG Baseline pictures with three components use the maximum number of Huffman tables per file, it is possible to add an arbitrary amount of dummy Tables at the end of the file by inserting Huffman table (DHT) segments containing encoded bits. One such Table can be identified easily during the decoding process. As the "defined" code words are not actually used, they do not necessarily need to be valid. Hence, it is possible to define up to 16 sets of 255 theroretically contradictory maximum length Huffman code words defining one 8-bit value each. In total, this allows storing $16 \cdot 255 \cdot 8 = 32640$ bits at the expense of $4099 \cdot 8 = 32792$ stored bits (see [12, p. 45]). Note that an additional four bytes are required for the marker (two bytes) and the length field (two bytes) per segment. As an arbitrary amount of dummy Tables with the same destination identifier can be inserted, the capacity of this approach is theoretically unlimited.

Similar to dummy Huffman tables, dummy quantization tables can be defined by inserting Quantization Table (DQT) segments. One such segment can store 8 bits for each of the 64 quantization table positions. This allows for storing $64 \cdot 8 = 512$ bits at the expense of $65 \cdot 8 = 520$ stored bits (see [12, p. 44]). Again, the four bytes of overhead for the marker (two bytes) and additional length field (two bytes) have to be accounted for once per segment. The capacity is, again, unlimited due to the theoretically unlimited amount of dummy Tables when using the same destination identifier.

3.3 Information hiding

As an alternative to bit-stream-based changes to signal the RoIs, classic information hiding approaches, especially steganographic ones, can be used. An overview of state-of-the art methods, of which we consider the widely used coefficient-based approaches, i.e., those which alter bits in the DCT domain, is given in [5]. As encrypted RoIs can typically be identified by the human eye, the main aim of using information hiding for signalling is not hiding the bits, but storing them within the image itself. Thus, hiding schemes like F5 [23] which are known to be vulnerable to attacks [7] are considered as well.

The approaches' capacities is not evaluated herein as it has been evaluated in the literature, e.g., [8] for coefficient-based information hiding. For JPEG images with one channel, i.e., grey-scale images, a capacity of 0.02 bits per non-zero AC coefficient has been reported. As we assume having three channels per image, it is safe to use the aforementioned capacity as a lower bound, requiring only to determine the average number of non-zero AC coefficients of the test data. Note that information hiding is not necessarily lossy as reversible approaches have been proposed (e.g., [16, 18]).

3.4 Length-preserving signalling

A method without overhead is the use of bits occupied by unused code words in the Huffman tables, i.e., code words

Method	Compliant	Lossless	Available	Length-preserving	Capacity (bits)
COM segment	✓	✓	✓	–	∞
APP segment	✓	✓	–	–	∞
EXIF data stripping	✓	✓*	–	✓	Variable
Dummy DHT	✓	✓	✓	–	∞
Dummy DQT	✓	✓	✓	–	∞
Steganographic (coefficients)	✓	–**	✓	Depends	Variable
Reuse of unused DHT entries	✓	✓	–	✓	Variable
DQT bit stealing	✓	Depends	–	✓	Variable
Data before first marker	–	✓	✓	–	∞
Data after last byte	–	✓	✓	–	∞

* Original EXIF data will be lost, ** Reversible techniques proposed (e.g., [16, 18])

Table 3: List of proposed methods for RoI signalling in JPEG pictures broken down by the aspects listed in Section 3

which are not used throughout the file. Although it is simple to find unused code words, even when e.g., decrypting, the number of unused code words may be very low or even zero, if the Huffman table only contains used code words or if there is no Huffman table to begin with. Even if there are unused code words, each of them only allows for storing 8 bits. Furthermore, as the number of unused code words varies from file to file, this method's capacity highly depends on the encoder which created the file and therefore has to be evaluated.

Another method of storing encoded RoI coordinates is by stealing bits from the quantization table(s), i.e., by modifying the bits of some quantization table entries, if there is a quantization table in the first place. There are two possibilities of doing so: One way is to change one bit at a time, starting at the high frequency entries of the chrominance quantization tables. After each modification, the JPEG file is decoded and compared to the version with unchanged quantization tables. Although this is computationally very expensive, it can also be done during the decoding process to find out which bits of the quantization tables were used. However, the capacity is highly dependent on the picture and possibly zero. Alternatively, if distortions are acceptable up to a certain degree, a fixed number of bits can be used, omitting the trial-and-error process described before. Although this allows for a higher capacity, it does so at the expense of picture quality, which has to be assessed.

3.5 Non-format-compliant signalling

A way to losslessly signal encoded RoI coordinates is to insert them at either the very beginning of the file, i.e., before the first marker, or at its end, i.e., after the last data byte. Adding data in this way is, however, not format compliant as the standard only allows for 0xFF fill bytes preceding each marker. In addition, in both cases, special care has to be taken in order to escape 0xFF payload bytes which would otherwise be interpreted as markers. Depending on how escaping is done, this may lead to additional overhead. As this method of signalling encoded RoI coordinates is not format compliant, most image viewers and editors will not be able to open files edited by it anymore.

4. EVALUATION

In order to evaluate the RoI signalling methods presented in Section 3 in combination with the coordinate encoding methods proposed in Section 2, we first evaluate each aspect separately and subsequently combine them. As a practical JPEG RoI encryption application we choose the encryption of people in pictures of surveillance cameras.

In total, eleven test sets are used – three indoors and eight outdoors sets. The three indoors data sets[1] with a total of 3271 pictures with a spatial resolution of $360 \cdot 288$ pixels each are courtesy of the EPSRC funded MOTINAS project (EP/D033772/1). The eight outdoors data sets[2] with a total of 67616 pictures with a spatial resolution of $640 \cdot 480$ pixels each are courtesy of EPSRC project GR/S98146. All data sets include ground truth for people's coordinates within each picture, which is subsequently used as set of RoIs to be encoded and signalled. RoIs which exceed one or more of the pictures' borders are omitted.

4.1 Encoded RoI bit length assessment

In order to perform coordinate encoding of the data sets' RoIs, we implemented the different encoding methods presented in Section 2 in Python, except for arithmetic encoding and JBIG compression, for which we used the Python implementation of David MacKay[3] and JBIG-KIT[4], respectively. As we restrict the coordinates' accuracy to iMCUs of $16 \cdot 16$ pixels size (see Section Section 2), we rounded the data sets' RoI coordinates so that all blocks containing an RoI were considered to be encrypted as a whole. Before actually encoding the rounded coordinates, they were translated into iMCU indices as explained in Section 2.2.

Tables 4 and 5 show the average number of bits per picture required to encode the RoIs of the indoors and the outdoors data sets, respectively. As the RoI count of the pictures has a significant impact on the number of bits required, the results are grouped by RoI count, considering only pictures from the data set with the stated number of RoIs. Note that pictures without, i.e., zero, RoIs are omitted as they are discussed separately in the second part of this Section.

It is clearly visible from the results of both data sets that entropy coding (in the right half of each Table) always im-

[1] ftp://motinas.elec.qmul.ac.uk/pub/av_people/

[2] http://groups.inf.ed.ac.uk/vision/BEHAVEDATA/INTERACTIONS/

[3] http://shedskin.googlecode.com/svn/trunk/examples/ac_encode.py

[4] http://www.cl.cam.ac.uk/~mgk25/jbigkit/

RoIs	Pictures	Bitmap	List	VList	DList	ACBitmap	ACList	ACVList	ACDList	JBitmap
1	1907	414.00	36.00	*30.94*	34.94	196.00	31.27	31.04	33.70	206.96
2	959	414.00	54.00	61.53	56.53	196.00	*52.07*	59.01	54.20	229.30
Non-0	2866	414.00	42.01	41.15	42.15	150.99	38.22	*40.38*	40.55	214.42

Table 4: Average number of bits required to encode the RoIs of each picture of the indoors data set with a given number of RoIs. The best, i.e., minimal, number of bits for each distinct picture subset is italicized

RoIs	Pictures	Bitmap	List	VList	DList	ACBitmap	ACList	ACVList	ACDList	JBitmap
1	1444	1200.00	44.00	*33.58*	37.58	138.32	35.64	34.04	36.05	212.34
2	3449	1200.00	66.00	64.72	55.80	241.83	58.78	60.92	*52.26*	231.46
3	2820	1200.00	88.00	92.81	81.97	255.98	80.66	85.31	*74.41*	238.52
4	1877	1200.00	110.00	135.89	106.16	333.66	105.88	121.25	*96.51*	245.66
5	10822	1200.00	132.00	166.46	135.98	393.94	128.51	150.00	*122.08*	260.68
6	814	1200.00	154.00	204.50	163.71	433.94	149.21	180.02	*144.71*	267.50
7	3	1200.00	176.00	232.67	168.67	430.67	177.00	208.00	*153.00*	266.67
8	2	1200.00	198.00	270.00	176.00	430.00	197.50	247.50	*158.50*	256.00
Non-0	21234	1200.00	108.38	129.92	107.54	329.75	103.34	117.70	*97.18*	248.64

Table 5: Average number of bits required to encode the RoIs of each picture of the outdoors data set with a given number of RoIs. The best, i.e., minimal, number of bits for each distinct picture subset is italicized

proves encoding efficiency, i.e., it reduces the number of bits, except in the case of only one RoI when using a list of variable-length coded indices ("VList"). Thus, implementing an entropy coding step following the actual coordinate encoding step should always be considered when encoding more than one RoI. In the case of a single RoI, variable-length coded indices ("VList") give the best results on average over all eleven test sets.

For a higher number of RoIs, the outdoors data sets (Table 5) allow for a more thorough analysis due to the data sets' widespread range of RoI counts. They clearly show that a differentially coded list of values which is entropy coded ("ACDList") always gives the best results. The higher the number of RoIs is, the higher the bit savings of this method are compared to all of the others besides "JBitmap". Note that there are only very few pictures with seven and eight RoIs, respectively, making the results only reliable for up to six RoIs. Nonetheless, averaging the number of bits spent over all pictures with RoIs (last line of Table 5) reveals that the "ACDList" encoding is optimal for data sets which contain a high number of pictures with more than one RoI. The maximum number of bits required for one list of RoIs over all data sets (not listed in the Table) is 219 bits.

Additionally considering the 959 pictures of the indoors data set (Table 4) containing two RoIs shows that an entropy coded list of indices ("ACList") yields a good performance as well, albeit only smaller by about two bits in this special case as compared to the "ACDList" encoding. Interestingly, the "ACVList" encoding shows the best overall performance over the complete indoors data sets, being 0.17 bits shorter than the "ACDList" encoding on average. This is due to the fact that the number of pictures in the indoors data set with one RoI is higher than the number of pictures with two RoIs and that the "ACVList" encoding requires the smallest number of bits for encoding one RoI as compared to all other entropy-coding-based encodings in the indoors data sets.

Surprisingly, JBIG compression performs significantly worse than most of our proposed approaches, which is mainly due to the large overhead caused by the JBIG file header. However, it is clearly visible from the outdoors data set in Table 5 that the JBIG based encoding requires fewer bits per additional RoI compared to all other approaches. While our "ACDList" approach requires on average 108.66 bits more for encoding six RoIs than it does for one RoI, "JBitmap" only requires 55.16 bits more. Thus, it is expected that JBIG compression outperforms our approaches for large numbers of RoIs.

As encoding zero RoIs, i.e., the fact that no RoIs are present, is independent of the data set used, both, Table 4 and 5, do not include pictures without RoIs. In order to assess the encoding methods' RoI encoding performance of pictures of this type in general, we used artificial images with different spatial dimensions, all of which had an aspect ratio, i.e., a width-to-height ratio, of 4:3 as is common in surveillance applications. Moreover, all image sizes were rounded to the next integer multiple of 16.

Figure 1 shows the number of bits required to encode zero RoIs for all proposed encodings with picture sizes ranging from $16 \cdot 16$ to $1920 \cdot 1440$ pixels in steps of 16 pixels in width. Although the aspect ratio is fixed (despite the small errors due to rounding), the X axis shows the square root of the image area, making the results applicable to arbitrary aspect ratios. The Y axis shows the required number of bits using a logarithmic scale.

The bit requirements for "VList" and "DList" as well as for their entropy coded variants, "ACVList" and "ACDList", are always constant, regardless of the picture's spatial dimensions, thus forming a combined line at two bits. This property makes the four encodings ideal for quasi all picture sizes, except for a size of $16 \cdot 16$, which we consider of having no practical use. Thus, each of the four encodings is recommended for encoding zero RoIs at all spatial picture dimensions used in practice.

Conversely, the "Bitmap" encoding's requirements in terms of bits increase quadratically with picture width (linearly with increasing picture area), making it inconvenient for

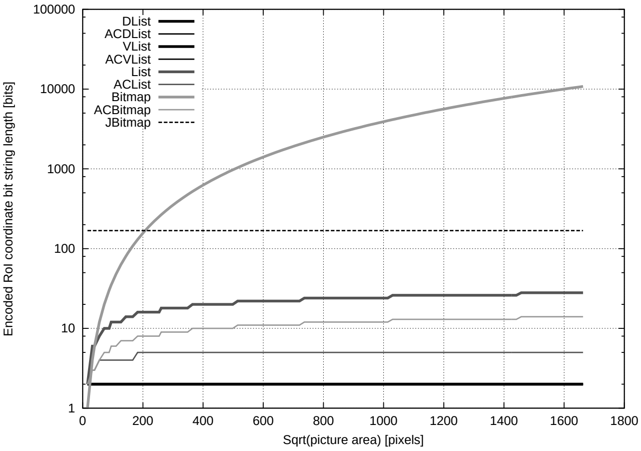

Figure 1: Number of bits required to encode zero RoIs for different spatial picture dimensions

practical use for pictures without RoIs. Note that entropy coding ("ACBitmap") reduces the required number of bits significantly, albeit still dependent on the picture's dimensions. The same is true for the "List" and "ACList" encodings, which are similarly inconvenient for encoding zero RoIs in practice.

Although the "JBitmap" encoding has a constant overhead, it is significantly larger (168 bits) than the overhead required by our Exponential-Golomb-based approaches described above (2 bits). Thus, it is not recommended to be used to encode zero RoIs.

As both, the indoors and the outdoors data sets, only cover a limited range of picture dimensions and RoI counts, we additionally assessed the bit length requirements of the proposed encodings using artificial test data sets. In order to artificially create RoIs that resemble real-world characteristics we use the following approach to create n_{RoI} RoIs:

- A random quadtree decomposition up to a maximum level l is created for a test image of dimensions $w \cdot h$, following the approach for creating uniformly distributed quadtree decompositions described in [17].
- n_{RoI} leaves are selected randomly from the set of all leaves (in case the number of leaves is less than n_{RoI}, a new quadtree is generated).
- For each selected leaf in the quadtree, a RoI is created with dimensions $q_w \cdot q_h$ where $q_w = \lceil f_w \cdot w \rceil$ with f_w chosen randomly such that $m \leq f_w < 1$. m denotes the minimum relative width and can be specified in $0 < m \leq 1$. q_h is chosen in an analogous manner.
- The position of the RoI is chosen randomly, ensuring that the RoI fits into the area covered by the leaf: Let (l_x, l_y) be the coordinates of the upper left-hand corner of the selected leaf with dimensions $l_w \cdot l_h$. The horizontal position of the RoI is determined as $\lfloor l_x + f_x \cdot (l_w - q_w) \rfloor$, with f_x chosen randomly and $0 \leq f_x < 1$. The vertical position of the RoI is determined in an analogous manner.

The parameters l and m can be used to tune the average size of the generated RoIs. In our test setup, we use $l = 3$ and $m = 0.2$ to simulate real-world RoIs.

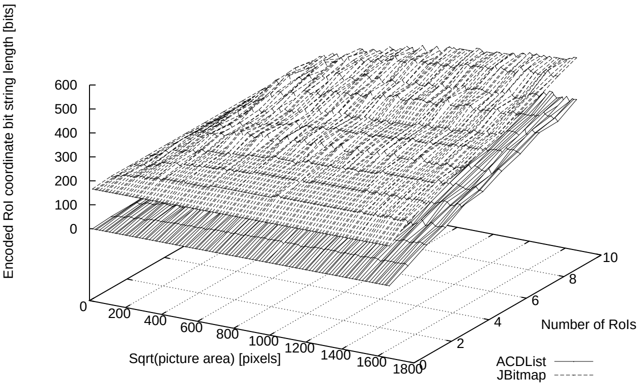

Figure 2: Number of bits required to encode a number of randomly generated, artificial RoIs for different spatial picture dimensions

Figure 2 depicts the number of bits required to encode $0 \leq n_{RoI} \leq 10$ randomly generated RoIs for various picture sizes (see above for details). Note that only "ACDList" and "JBitmap" are depicted for comparison for the sake of better graphical representation. All other encodings (not depicted) perform worse except for very small picture sizes which we do not consider being practical, as discussed above, with one exception described below. It is clearly visible that "JBitmap" does not outperform "ACDList" at any picture size or RoI count depicted. This is due to the former's large overhead, confirming that JBIG may only be suitable for a very large number of RoIs. As "ACVList" performed better than "ACDList" for a small number of RoIs, the two encodings are compared for $0 \leq n_{RoI} \leq 10$ randomly generated RoIs for various picture sizes. Figure 3 depicts the number of bits which are required by the "ACVList" encoding in addition to "ACDList"'s for any given configuration. Note that the X axis starts at 16 as a picture size of 0 is not practically useful. As in the real-world data sets, "ACDList" outperforms "ACVList" for a large number of RoIs. However, in some configurations with one or two RoIs, "ACDList" outperforms "ACVList" by a few bits. Nonetheless, the "ACDList" encoding is clearly the encoding of choice, when the number of RoIs is not known in advance and potentially large.

4.2 JPEG picture capacity assessment

As some of the signalling approaches described in Section 3 have an unknown embedding capacity, we evaluate the latter for our test data sets. The approaches considered herein are the reuse of unused DHT entries, steganographic approaches and DQT bit stealing as described in Section 3. To evaluate the embedding capacity when reusing unused DHT entries, we count the latter in the outdoors data sets' JPEG files, whose JPEG quality varies between approximately 95 and 100%. Those pictures which have a quality of approximately 100% do not have unused DHT entries at all. Conversely, the pictures which have a quality of approximately 95% allow using 214 entries on average with one byte of capacity each, i.e., 1712 bits in total. The minimum and maximum number of unused entries is 191 (corresponding to 1518 bits) and 343 (2744 bits), respectively, which, in our opinion, is surprisingly high.

As the outdoors data sets' approximate JPEG quality is

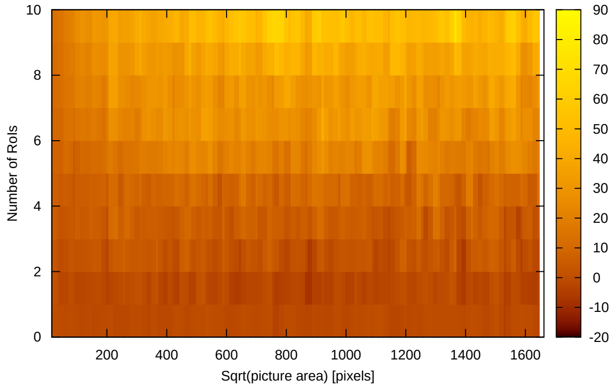

Figure 3: Additional number of bits required for the "ACVList" encoding to encode a number of randomly generated, artificial RoIs for different spatial picture dimensions compared to the "ACDList" encoding

within a very limited range, an evaluation spanning a wider range of JPEG quality values is necessary. As the indoors data sets are already compressed using a different compressor, applying JPEG compression would also recompress the existing artifacts, yielding results which are not representative. Thus, we use images from the LIVE data base [21] and compress them with the standard JPEG encoder with quality values ranging from 0 to 100% in 5% steps.

The results are depicted in Figure 4 and show that low JPEG quality values allow for a higher embedding capacity than high quality values. This is due to the standard JPEG encoder using a fixed DHT, making unused entries more likely for lower quality due to the stronger quantization and therefore longer runs with lower absolute coefficient values. Note that the embedding capacity for a quality value of 95% is approximately 1800 bits on average with a minimum value of about 1600 bits, which matches the outdoors data sets' capacity within a small bound.

To evaluate the embedding capacity of steganographic approaches, we use the approximation described in Section 3 estimating the embedding capacity as 0.02 bits per non-zero AC coefficient. Similar to the embedding method explained above, we count the non-zero AC coefficients in all JPEG images of the outdoors data sets. After omitting one image which is all black, we find an average number of about 195117 non-zero AC coefficients per file with a minimum and maximum of 49440 and 246679, respectively. This corresponds to an embedding capacity of about 3902 bits on average with a minimum of 988 bits.

Again, we also count the number of non-zero AC coefficients of the images from the LIVE data base with different JPEG quality values to cover a wider range of the latter. As can be seen in Figure 5, the number of non-zero AC coefficients and therefore the embedding capacity increases quasi linearly for increasing low JPEG quality values and exponentially with increasing high JPEG quality value (note the logarithmic Y scale). For a JPEG quality value of 95%, the embedding capacity is approximately 3000 bits on average with a minimum of about 2000 bits, which differs from the outdoors data sets' capacity, but is within the same order of magnitude.

Finally, we evaluate the DQT bit stealing approach. In order to simulate a worst case bit-stealing scenario, we flip n bits of each 8-bit DQT entry from indices i_1 to i_2 in zigzag order, i.e., in bit stream order of both, luminance and chrominance DQT.

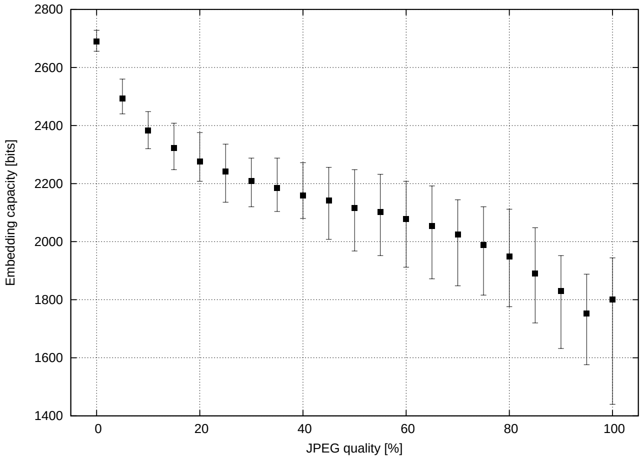

Figure 4: Average embedding capacity of the unused DHT entry reuse approach for different JPEG quality values for the pictures of the LIVE data base [21]. The bars indicate the minimum and maximum capacity for each quality value, respectively

To find suitable values for n, i_1 and i_2, we take a picture data set, decode each picture and then compare it with a decoded version with flipped QT Table entries for all possible values of n, i_1 and i_2. In order to assess the difference between the original and the modified picture, we measure the PSNR value between the two.

As the number of possible combinations of n, i_1 and i_2 is large, we evaluated them exhaustively on a smaller test set – the LIVE data base [21]. Using the JPEG reference encoder, we created Baseline JPEG images with default settings and 50, 75, 95 and 100% quality from the original, i.e., uncompressed, images.

Figure 6 shows the embedding capacity (in terms of total stolen bits, Y axis) over all images of a given JPEG quality so that the distortion of no image exceeds the depicted PSNR value (X axis). Note that the JPEG quality influences the embedding capacity significantly, as does the desired maximum distortion.

Surprisingly, the total capacity is very high, considering that changes of the DQT potentially influence all blocks of a picture. Depending on the desired target distortion, it is possible to embed several hundred bits.

Note that 100% quality does not allow embedding one bit so that no picture exceeds a distortion of 50dB. The same is true for all JPEG quality values when no distortion (∞ dB) is desired. Thus, this approach cannot be used for lossless embedding.

Attempting to verify these results for the outdoors data sets, we split the data sets into pictures with approximately 95% and 100% JPEG quality, respectively, using the obtained settings for a target quality of 35dB. Surprisingly, every picture in both sets exceeds a quality of 50dB compared to its unmodified version, indicating that the embedding capacity for a given target quality is highly dependent on the pictures themselves. Due to the lack of freely available and

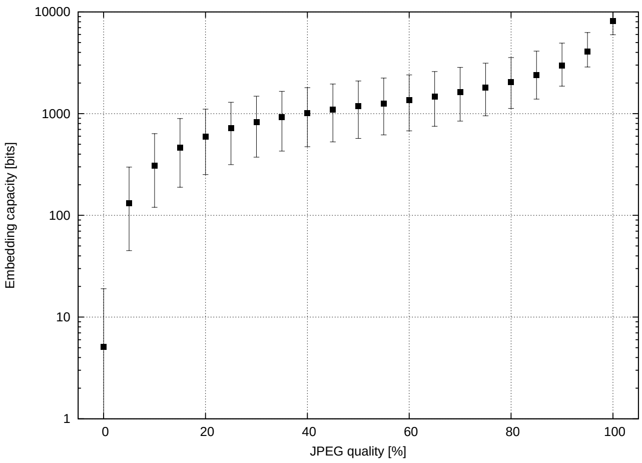

Figure 5: Average embedding capacity of steganographic approaches for different JPEG quality values for the pictures of the LIVE data base [21]. The bars indicate the minimum and maximum capacity for each quality value, respectively

practically relevant data sets, a thorough examination of this method with more pictures of different characteristics remains future work.

4.3 Combined encoding and signalling

Combining the results of the previous Sections, we subsequently evaluate the feasibility of the combined use of the proposed encoding and signalling methods in order to simulate the actual storage of RoI coordinates in the corresponding JPEG files. Due to the lack of freely available JPEG-encoded data sets with RoI ground truth, only the outdoors data sets can be assessed in this Section. Although this allows no general conclusions regarding the usefulness of the proposed approaches, it is possible to determine possible combinations of signalling and encoding methods suitable for the outdoors data sets, which cover a significant portion of practically relevant pictures and RoI counts for surveillance and encryption applications.

As the average number of bits for encoding RoI coordinates in the outdoors data sets is smallest when using our proposed "ACDList" approach (see Section 4.1), we consider the latter to be appropriate for encoding all RoIs. This choice is further supported by the fact that our "ACDList" approach is one of the few approaches which allows signalling the absence of RoIs by just two bits. Note that for data sets where zero or one RoI(s) are dominant, our "ACVList" approach allows using fewer bits.

Subsequently, we determine which of the signalling approaches described in Section 3 are able to provide enough embedding capacity to store the "ACDList"-encoded RoI coordinates for the outdoors data sets. Trivially, all approaches which offer infinite capacity can be used to signal the RoIs which require a maximum number of 219 bits with our proposed "ACDList" encoding. As format-compliant approaches are in general preferred over non-format-compliant ones, we suggest using COM segments as their overhead is smallest as compared to all other methods which offer infinite capacity (see Section 3).

The reuse of unused DHT entries allows for a largely sufficient minimum capacity of 1518 bits in all pictures with approximately 95% JPEG quality, allowing for lossless and length-preserving RoI signalling. To our knowledge, this is the first time that such an approach has been proposed. However, the pictures with approximately 100% JPEG quality do not allow storing a single bit using this method, making it unusable in this scenario. Consequently, we suggest using this method instead of others whenever possible as its capacity is sufficient to store large numbers of RoIs without quality loss and change of file size.

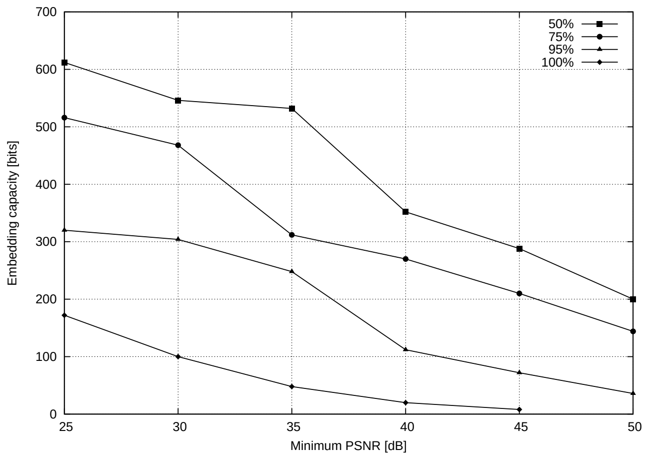

Figure 6: Embedding capacity of the DQT bit stealing approach for different JPEG quality values and maximum target distortions for the pictures of the LIVE data base [21]

If quality loss is acceptable, classical steganographic methods allow for a high capacity when using a generalized estimation of the embedding capacity. It is notable that the minimum estimated capacity of these approaches is lower (988 bits) than the minimum capacity provided by the reuse of unused DHT entries, if the latter is available (1518 bits). However, steganographic approaches can be used on practically any picture, making it the method of choice when quality loss is acceptable. Note that the actual capacity highly depends on the method used as well as on the desired distortion, which is outside of the scope of this paper. We refer to the available literature [5] for further details.

Finally, our proposed approach which steals bits from the DQT also allows signalling RoIs, although its capacity is limited and highly dependent on the desired quality in terms of PSNR as well as on the pictures' characteristics. As described in Section 4.2, further evaluations are required in order to determine the usefulness of this method. In general, it can be noted that its capacity is surprisingly high in all tested cases, but limited for large numbers of RoIs as well as for pictures with large spatial dimensions. As changing the DQTs typically influences all blocks of a picture, as opposed to most steganographic approaches, which operate on single blocks, steganographic approaches are recommended at this point, with our approach being an option to be considered as soon as it has been evaluated more thoroughly.

5. CONCLUSION

We proposed and evaluated several methods to encode and signal RoIs in JPEG images. Using a number of data sets, we determined that our proposed arithmetically coded differential lists of iMCU indices are superior to all other evaluated RoI coordinate encoding methods for a large range of RoI counts, outperforming JBIG in this special use case. Furthermore, we showed that using JPEG comment segments to store the encoded RoI coordinates causes the lowest overhead, if the file size is allowed to change. For scenarios which require length-preservation, we proposed a new method which reuses unused Huffman table entries. Although it is not always available, it allows for lossless and length-preserving signalling, if it is available. Finally, we showed that using quantization table bits allows for RoI signalling as well, although further tests are required in order to determine the general restrictions of this method.

6. ACKNOWLEDGMENTS

The authors would like to thank Matthew Sorell for his valuable ideas on which several of the RoI signalling methods are based. This work is supported by FFG Bridge project 832082.

7. REFERENCES

[1] M. Adams. The JPEG-2000 still image compression standard. ISO/IEC JTC 1/SC 29/WG 1 N 2412, Sept. 2001.

[2] T. E. Boult. PICO: Privacy through invertible cryptographic obscuration. In *IEEE/NFS Workshop on Computer Vision for Interactive and Intelligent Environments*, pages 27–38, Lexington, KY, USA, Nov. 2005.

[3] P. Carrillo, H. Kalva, and S. Magliveras. Compression Independent Reversible Encryption for Privacy in Video Surveillance. *EURASIP Journal on Information Security*, 2009:1–13, Jan. 2009.

[4] T. Chattopadhyay and A. Pal. Watermarking H.264 video, Nov. 2007. Available online: `http://www.dspdesignline.com/202805492`.

[5] A. Cheddad, J. Condell, K. Curran, and P. M. Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal Processing*, 90(3):727–752, 2010.

[6] CIPA. DC-008-2010. Exchangeable image file format for digital still cameras: Exif Version 2.3, Apr. 2010. Also published as JIETA: CP-3451B.

[7] J. Fridrich, M. Goljan, and D. Hogea. Steganalysis of JPEG Images: Breaking the F5 Algorithm. In F. A. Petitcolas, editor, *Information Hiding*, volume 2578 of *Lecture Notes in Computer Science*, pages 310–323. Springer Berlin Heidelberg, 2003.

[8] J. Fridrich, T. Pevný, and J. Kodovský. Statistically Undetectable JPEG Steganography: Dead Ends, Challenges, and Opportunities. In *Proceedings of the 9th Workshop on Multimedia & Security*, MM&Sec '07, pages 3–14, New York, NY, USA, 2007. ACM.

[9] ITU. T.82: Information technology – Coded representation of picture and audio information – Progressive bi-level image compression, Mar. 1993.

[10] ITU. T.85: Application profile for Recommendation T.82 – Progressive bi-level image compression (JBIG coding scheme) for facsimile apparatus, Aug. 1995.

[11] ITU-T H.264. Advanced video coding for generic audiovisual services, Nov. 2007.

[12] ITU-T T.81. Digital compression and coding of continuous-tone still images — requirements and guidelines, Sept. 1992. Also published as ISO/IEC IS 10918-1.

[13] C. Kailasanathan. Compression performance of JPEG encryption scheme. In *Proceedings of the 14th International IEEE Conference on Digital Signal Processing, DSP '02*, July 2002.

[14] D. A. Kerr. Chrominance Subsampling in Digital Images. `http://dougkerr.net/pumpkin/articles/Subsampling.pdf`, Jan. 2012.

[15] M. I. Khan, V. Jeoti, and A. S. Malik. On perceptual encryption: Variants of DCT block scrambling scheme for JPEG compressed images. In T.-H. Kim, S. K. Pal, W. I. Grosky, N. Pissinou, T. K. Shih, and D. Slezak, editors, *FGIT-SIP/MulGraB*, volume 123 of *Communications in Computer and Information Science*, pages 212–223. Springer, 2010.

[16] W.-C. Kuo, S.-H. Kuo, and L.-C. Wuu. High Embedding Reversible Data Hiding Scheme for JPEG. In *Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, pages 74–77, oct. 2010.

[17] R. Kutil and D. Engel. Methods for the anisotropic wavelet packet transform. *Applied and Computational Harmonic Analysis*, 25(3):295–314, 2008.

[18] C.-C. Lin and P.-F. Shiu. DCT-based reversible data hiding scheme. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '09, pages 327–335, New York, NY, USA, 2009. ACM.

[19] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July 2003.

[20] W. Pennebaker and J. Mitchell. *JPEG – Still image compression standard*. Van Nostrand Reinhold, New York, 1993.

[21] H. R. Sheikh, M. F. Sabir, and A. C. Bovik. A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE Transactions on Image Processing*, 15(11):3440–3451, Nov. 2006.

[22] A. Unterweger and A. Uhl. Length-preserving Bit-stream-based JPEG Encryption. In *MM&Sec'12: Proceedings of the 14th ACM Multimedia and Security Workshop*, pages 85–89. ACM, Sept. 2012.

[23] A. Westfeld. High capacity despite better steganalysis, F5 - a steganographic algorithm. In *Proceedings of the 4th Information Hiding Workshop '01*, Portland, OR, USA, Apr. 2001.

[24] I. Witten, R. Neal, and J. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.