MASTER THESIS

Automated Detection of Encrypted RoIs in JPEG2000

prepared at

Salzburg University of Applied Sciences

Master Degree Program

Information Technology & Systems Management

submitted by:

Andreas Weitlaner, BSc

Head of Faculty:	FH-Prof. DI Dr. Gerhard Jöchtl
Supervisor:	FH-Prof. DI Mag. Dr. Dominik Engel

Registration Number: 1110581037

Salzburg, August 2013

Affidavit

I hereby declare that I wrote this thesis on my own and without the use of any other than the cited sources and tools and all explanations that I copied directly or in their sense are marked as such, as well as that the thesis has not yet been handed in neither in this nor in equal form at any other official commission.

Salzburg, 7th August 2013

Andreas Weitlaner, BSc	Registration Number

Details

First Name, Surname:	Andreas Weitlaner
University:	Salzburg University of Applied Sciences
Master Degree Program:	Information Technology and Systems Management
Title of Thesis:	Automated Detection of Encrypted RoIs in JPEG2000
Academic Supervisor:	FH-Prof. DI Mag. Dr. Dominik Engel

Keywords

1^{st} Keyword:	JPEG2000
2^{nd} Keyword:	Automated RoI detection
3^{rd} Keyword:	Region of Interest
4^{th} Keyword:	Scrambling / Encryption
5^{th} Keyword:	Data hiding / Signaling

Make everything as simple as possible, but not simpler.

Albert Einstein

Acknowledgements

First of all, I would like to thank my Master's Thesis advisor
FH-Prof. DI Mag. Dr. Dominik Engel for his support and guidance.
Our meetings and discussions made this thesis possible.
I am especially grateful to my girlfriend for her patience
and insightful comments when I was struggling.
Finally, thanks to my family for always standing by me.

This work has been partly supported by the
Forschungsförderungsgesellschaft GmbH under FFG Bridge project 832082.
Furthermore, this work has been partly supported by the
Austrian Marshall Plan Foundation which offered me the
unique opportunity to spend a semester at the Polytechnic Institute
of New York University.

Abstract

With the wide use of video surveillance systems in public and private spaces the question of individual privacy concerns receives increasing attention [1, 2]. It is out of question that the recording of a person's visual appearance poses a threat to personal privacy. To preserve personal privacy, privacy-preserving video surveillance systems have been proposed since 2006, with the basic idea to protect the facial area of people being recorded by some cryptographic means [3]. The goal of the proposed research project is to improve and extend existing RoI encryption techniques focusing on encryption applied to the media bitstream, specifically targeting the scalable format JPEG2000 [4, 5]. Focus of the project is on improving real world applicability of JPEG2000 RoI encryption by creating a format-compliant representation of the privacy-protected bitstream.

An approach that allows determining the encrypted regions automatically will be developed. For this purpose, we will investigate on the one hand the completely automatic detection of encrypted regions without any additional information and on the other hand we use data hiding techniques, to insert additional information into the JPEG2000 bitstream, for locating the RoIs. All approaches will be evaluated regarding computational demand, impact on image quality, format compliance and real-world feasibility.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the increasing usage of video surveillance systems in public and private spaces, the question of how to deal with privacy concerns of people being recorded receives increasing attention [1, 2]. It is out of the question that the recording of a person's visual appearance poses a threat to personal privacy. In order to account for these concerns, privacy-preserving video surveillance systems need to be developed. Besides the possible application of automated tracking and recognition systems, the unlimited observation of the actions of any recorded person by human security personnel is another major issue raising privacy concerns. It is important to notice that for most surveillance-relevant application scenarios, it is not required that the identity of persons being recorded is revealed at first. Privacy-preserving video surveillance systems have been proposed since 2006: The basic idea is to protect the facial area of people being recorded by some cryptographic means. Most proposals employ some sort of region-of-interest (RoI) encryption technique for this task [16, 17]. The goal of the proposed research project is to improve and extend existing RoI encryption techniques focusing on encryption applied to the media bitstream, specifically targeting the scalable format JPEG2000 [4, 5]. The focus of the project is to improve real world applicability of JPEG2000 RoI encryption by creating a format-compliant representation of the privacy-protected bitstream. As outlined above, the principal approach has significant application potential but techniques developed so far suffer from several shortcomings in terms of practical feasibility. To some extent this is due to the fact that meta- and key-data, required to decrypt the encrypted image regions, need to be stored separately from the surveillance video data. This is due to the fact, as outlined in Chapter 3, that additional data such as the cipher-key (used to decrypt the bitstream parts), the position and length of encrypted bitstream sections, the encryption-counters, etc., must be stored to successfully detect and decrypt the encrypted image regions. An approach will be developed that allows determining the encrypted regions without additional meta data. For this purpose, we will investigate the efficient auto-

mated detection of encrypted regions in the JPEG2000 bitstream. A second angle to be investigated is the use of data hiding techniques to store the specification of the encrypted regions in the unencrypted parts of the image. All approaches will be evaluated regarding computational demands, impact on image quality and real-world feasibility.

1.1 State of the Art

The aim of this section is to give an overview of the latest research results and findings in the field of privacy-preserving video surveillance systems. It specifically targets the scalable format JPEG2000, which is used by this work to evaluate different embedding and automated RoI detection approaches. JPEG2000 is the latest image compression standard developed by the Joint Photographic Experts Group (JPEG) [9, 18]. This standard has been developed because existing standards lack efficiency and flexibility [18]. Furthermore, JPEG2000 offers among other features the ability to efficiently handle RoIs [9]. Hence, the JPEG2000 standard offers the possibility to encode- and decode certain image regions differently (e.g., the facial area of people being recorded by a video surveillance system). However, in the development of the JPEG2000 standard, decisions for not storing the RoI position explicitly in the resulting media bitstream were made. This decision leads to the fact that, without decoding the image or adding additional information, it is not possible to extract the position of the RoI (for further details see Chapter 2).
Therefore methods such as the one described by Hämmerle-Uhl et al. [19] and implemented by Stubhann [5], where parts of the media bitstream are encrypted to preserve the privacy of the persons recorded, need an additional file which stores the actual position of the encrypted bitstream parts for decrypting the encrypted parts of the bitstream. Stubhann implemented and evaluated three different privacy-preserving JPEG2000 RoI encryption methods, which are applied at codestream level, after applying the JPEG2000 encoder to the input image (see Section 3.1). For detecting the codestream sections representing the RoI (e.g., the facial area), Stubhann used the following RoI detection approaches: RoI Max-Shift, Codeblock, and Tiles (see Section 3.3 for further details). Although JPSEC offers the possibility to encrypt selective/partial image regions by defining a Zone of Influence (ZoI, for further details see Subsection 2.4.7), this work is based on the format compliant JPEG2000 encryption, proposed by Hämmerle-Uhl et al. [19].
Nevertheless, JPEG2000 is capable of decoding RoIs with the Max-Shift method, for example (for a more detailed explanation of this and other RoI coding methods see [9]). However, this method is not suitable for detecting the encrypted RoIs without decoding the bitstream. Because of this circumstance and the lack of an extra field in the JPEG2000

header to store all the RoI specific information, a different solutions for storing the RoI data needs to be found. One such solution could be data hiding, which offers the ability to store arbitrary data in the image without a perceptual impact. With data hiding, it is possible to hide messages in an image. These messages are only able to be read by someone who knows where to look for them. Some implementations for hiding data in a JPEG2000 image have already been published (for more detailed information about data hiding, see Wayner's book [20] or for a JPEG2000 data hiding implementation, see Zhang et al. [21]). However, all these data hiding methods lack the feasibility in conjunction with Stubhann's implementation of a JPEG2000 bitstream RoI encryption, which follows an approach proposed by Hämmerle-Uhl et al. [19]. This is due to the fact that Hämmerle-Uhl's media bitstream encryption method is invoked after encoding and finalizing image compression, and it is not possible to combine it with the outlined data hiding methods proposed so far, because all of them must to be executed while encoding the image. This needs to be done this way because the proposed methods need to identify parts of the image in which it is possible to hide the information, without degrading the image quality nor causing any perceptual impact.

Another technique to embed information into digital images is watermarking, which has been studied for copyright protection extensively in recent years [22, 23, 24, 25]. Most of the proposed watermarking methods are, along with the steganography methods, invoked while encoding the digital image and therefore not applicable in conjunction with the privacy-preserving method proposed by Hämmerle-Uhl et al. [19] and Stubhann's [5] implementation. Some research has been done on watermarking on bitstream level, such as the proposed method by Katsutoshi et al. [26] which embeds data into a JPEG2000 bitstream by overwriting certain parts of the bitstream. This method has the advantage of detecting the embedded data very efficiently, as no data needs be decoded for extracting the hidden data in the JPEG2000 bitstream. However, one drawback of this method, which will be further evaluated in this master thesis, is the image quality degradation. The image quality degradation is based on the fact that certain parts of the bitstream need be overwritten for embedding the encryption specification.

Nevertheless, extensive studies have been carried out in the fields of encrypting parts of the JPEG2000 bitstream [16, 27, 5, 4, 19] and data embedding methods [26, 24, 25]. No methods for automatically detecting the encrypted parts of the bitstream have been proposed thus far. The objective of this master thesis is to evaluate different methods of embedding data or automatically detecting encrypted parts of the JPEG2000 bitstream because techniques developed thus far suffer from several shortcomings and limitations in terms of practical feasibility.

1.2 Research question and objectives

Based on the situation outlined in the introduction, the main objective of this master thesis is to investigate the **automatic detection of encrypted RoIs in the JPEG2000 media bitstream**. Therefore, the main objective is subdivided into two approaches, (1) the automatic detection of RoIs by using data hiding techniques to store specification of the encrypted regions in the unencrypted parts of the image and (2) the detection of RoIs without any additional information.
All investigated approaches are based on Stubhann's implementation of a JPEG2000 RoI bitstream encryption, which follows an approach proposed by Hämmerle-Uhl et al. [19]. To be precise, the "Max-Shift RoI encryption" method (see Section 3.3) is used to evaluate the results regarding computational demands, impact on image quality, JPEG200 format-compliance and real-world feasibility [5].

1. **Data hiding techniques:**
    - Embed data into JPEG2000 header fields: Embed encryption specification into the JPEG2000 COM-segment.
    - Non-format compliant data embedding: Embed encryption specification either prior to the JPEG2000 SOC-marker or after its EOC-marker.
    - Length-preserving data embedding: Embed encryption specification by replacing JPEG2000 image coefficients.
2. **Automatic detection of RoIs, without additional information:**
    - Feasibility of automatic detecting RoI by using only the JPEG2000 bitstream: Without any decoding or additional data (no data hiding), it is investigated whether or not it is possible to extract the position of the encrypted data in the JPEG2000 bitstream.
    - Decode bitstream up to inverse wavelet transformation: Investigate feasibility of detecting RoIs by wavelets.
    - Completely decode the image: After completely decoding the image, the encrypted RoI is extracted by entropy, variance or edge detection methods.

1.3 Thesis Structure

The remainder of this master thesis is structured as follows: The first few chapters give theoretical background knowledge about JPEG2000, media bitstream encryption based on JPEG2000, data hiding and methods to automatically detect a Region-of-Interests in the JPEG2000 media bitstream. Therefore, this part comprises multiple chapters (see Chapters 2, 3, 4 and 5) that provide theoretical knowledge and references to further readings. The second part (Chapter 6) describes the implementation of different RoI detection approaches (embedding the encryption specification and automated RoI detection).
Finally, the third part discusses the evaluation and comparison of the results of the proposed methods according to the theoretical assumptions we obtained from the theoretical analysis (see Chapter 7). This part concludes with Chapter 8, which gives a summary and some recommendations.

Chapter 2

JPEG2000 Standard

As stated by the Joint Photographic Expert Group (JPEG) [28]:

> *JPEG2000 is a new image coding system that uses state-of-the-art compression techniques based on wavelets technology. Its architecture should lend itself to a wide range of uses from portable digital cameras to advanced pre-press, medical imaging and other key sectors.*

This chapter provides an overview of the compression standard JPEG2000.

2.1 JPEG2000 Introduction

JPEG2000 is the latest compression standard for still images, created by the Joint Photographic Experts Group (JPEG) and coordinated by the Joint Technical Committee on Information Technology of the International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC) [28].
With the continual expansion of multimedia- and Internet-applications, the needs and requirements of the technologies used grew and evolved. Therefore, in March 1997, a new call for contributions to develop a new standard for the compression of still images, the JPEG2000 standard, was launched [29]. JPEG2000 was developed with the objective to overcome the shortcomings of the predominant JPEG standard, which was released in 1992 and is still one of the most widely used compression standard for still images [18]. JPEG2000 provides a whole new way of interacting with compressed images in a scalable and interoperable fashion. Some of JPEG2000's numerous advantages and features over its predecessor JPEG [30, 31], cited by Taubman et al. [18] and Skodras et. al [29], are:

- **Superior low bit-rate performance:** This standard offers superior performance in comparison to the prevailing image compression standard JPEG at low bit-rates (e.g., below 0.25 bpp (bits per pixel) for highly detailed grey-scale images). This significantly improved low bit-rate performance is achieved without sacrificing per-

formance on the rest of the rate-distortion spectrum. Examples of applications that need this feature include network image transmission and remote sensing. This has been the feature in the JPEG2000 standardization process with the highest priority.

- **Continuous-tone and bi-level compression:** This coding standard is capable of compressing both continuous-tone (e.g., photographs or television images that have a virtually unlimited range of color or shades of grays) and bi-level images (a digital image that has only two possible values for each pixel, e.g., black and white). Another development goal for this new standard was to strive to achieve continuous-tone and bi-level compression with similar system resources. The JPEG2000 system, if implemented correctly, is capable of compressing and decompressing images with various dynamic ranges (i.e. 1 bit to 16 bit) and multiple components. Examples of applications that can use this feature include compound documents with images and text, medical images with annotation overlays, and graphic and computer generated images with binary and near to binary regions and alpha and transparency planes.

- **Lossless and lossy compression:** The JPEG2000 compression standard provides lossless and lossy compression. Lossy compression is achieved by applying the Daubechies 9-tap/7-tap irreversible wavelet transformation and the lossless compression is achieved by applying the Le Gall 5-tap/3-tap reversible wavelet transformation. Examples of applications that can use this feature include medical images, where loss is not always tolerated, image archival applications, where the highest quality is vital for preservation but not necessary for display, network applications that supply devices with different capabilities and resources, and pre-press imagery.

- **Progressive transmission by pixel accuracy and resolution:** Progressive transmission that allows images to be reconstructed with increasing pixel accuracy or spatial resolution is essential for many applications. This feature allows the reconstruction of images with different resolutions and pixel accuracy, as needed or desired, for different target devices. Examples of applications include the World Wide Web, which uses this feature to steadily increase image quality while loading the image from the internet or image archival applications and printers.

- **Region-of-interest (RoI) coding:** Often there are parts of an image that are more important than others (e.g., facial area is generally of greater importance than the background). This feature allows users to define certain RoIs in the image to be coded and transmitted in a better quality and less distortion than the rest of the image. Furthermore, this feature allows random codestream processing which could allow

operations such as rotation, translation, filtering, feature extraction and scaling.

- **Robustness to bit-errors:** For wireless communication, for instance, it is desirable to offer robustness to bit-errors, otherwise an effective communication would not be possible. Therefore, while designing the codestream, robustness needs to be considered a major part. Portions of the codestream may be more important than others in determining decoded image quality. Proper design of the codestream can aid subsequent error correction systems in alleviating catastrophic decoding failures.
- **Open architecture:** JPEG2000 has been developed as an open architecture, which has the advantage to allow a better interoperability with different image types and applications. Due to that fact, it is now possible to implement a decoder with core functionalities, which is able to parse all JPEG2000 format-compliant codestreams.
- **Protective image security:** Protection of a digital image can be achieved by means of different approaches, such as watermarking, labeling, stamping, or encryption. The open architecture of the JPEG2000 standard makes it possible to integrate such protection techniques into the JPEG2000 coder very easily.

2.2 Basic Architecture

Figure 2.1 depicts the basic operations a JPEG2000 encoder needs to fulfill while processing image samples. The bottom of the block diagram depicts the main processing steps, which are explained in more detail in this section. However, before proceeding with the details of each processing step, it should be mentioned that the standard encoding/decoding works on tiles. The tile-components are the basic units, which are treated by the coder as completely distinct images, and therefore compressed independently. Tiling reduces memory requirements, and since tiles are encoded and decoded independently, they can be used for decoding specific parts of an image instead of the whole image. An image can either be divided into one tile, which covers the whole image, or into multiple non-overlapping rectangular blocks (tiles) of equal size (expect those at the image borders) [6, 9, 29, 18]. For further details, reference is made to the book "JPEG2000 Image Compression Fundamentals, Standards and Practice" by Taubman and Marcellin [9].

2.2.1 Pre-processing

In the first stage, pre-processing is performed. JPEG2000 pre-processing is sub-divided into three stages, as shown in Figure 2.2. All these steps must be performed, otherwise the discrete wavelet transformation, which represents the next step in processing a JPEG2000

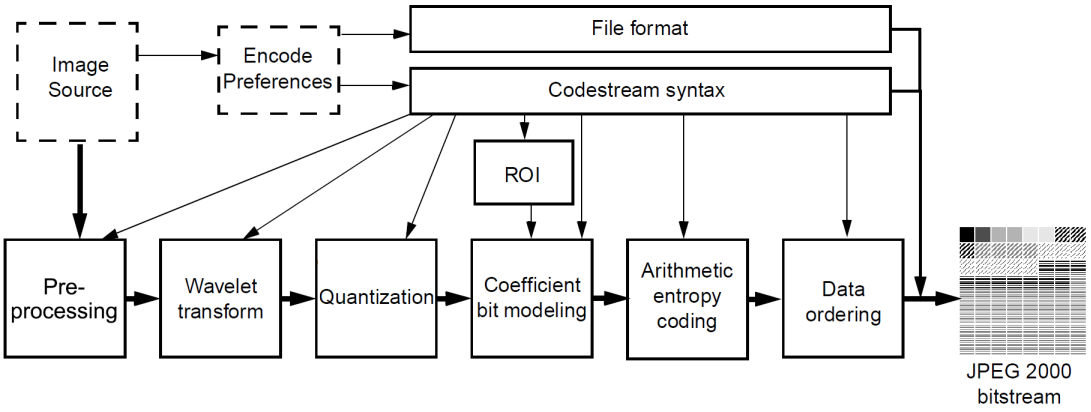


Figure 2.1: Block Diagram – JPEG2000 Encoder [6]

Figure 2.2: Pre-processing Substages – JPEG2000 Encoder [7]

image, would not work properly [7]. As mentioned in the introduction to this section, tiling needs to be performed to reduce memory demand; otherwise the coding of large images might not work or might take too long.

The second pre-processing stage ensures that the nominal dynamic range is centered around zero. This stage, called DC level shifting, needs to be performed prior to the discrete wavelet transformation (DWT), as JPEG2000 uses high-pass filtering while encoding an image. DC level shifting is performed on the samples of components that are unsigned only [18].

The third pre-processing stage deals with the component transformation, which provides decorrelation among image components. This pre-processing step improves the compression and allows for visually relevant quantization [6]. JPEG2000 supports up to 2^{14} components, and each component consists of a matrix of samples representing the luminosity of the component at that point [7]. Color images are most commonly represented in RGB format, which leads to three components (one for each color; red, green, blue). However, since Y (luminance), C_r (blue-difference chrominance components) and C_b (red-difference chrominance components) color components are less statistically dependent than RGB color components, they independently compress better. Therefore, the JPEG2000 standard has decided to convert RGB data into YC_rC_b data for a better transformation performance [7]. The JPEG2000 standard supports two component transformations, one that can be used for irreversible component transformation (ICT, lossy coding, see Figure 2.3) and one for reversible component transformation (RCT, lossless coding) [18].

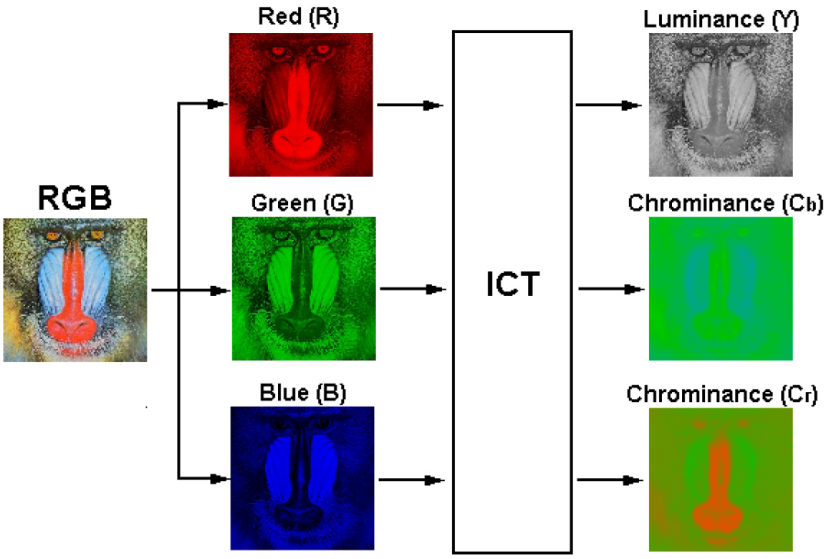


Figure 2.3: Irreversible component transformation (ICT) of a baboon image [7]

2.2.2 Wavelet Transform

During the wavelet transformation, each tile component is passed recursively through low pass and high pass wavelet filters. This procedure enables an intra-component decorrelation that concentrates the image information in a small and very localized area. Furthermore, it enables a multi-resolution image representation, which is one of the JPEG2000 core features [11].

There are two different wavelet transforms defined by the JPEG2000 standard, one for lossy and the other for lossless compression. Both provide lower resolution images and spatial decorrelation of the image to improve compression. The Daubechies 9-tap/7-tap irreversible wavelet transformation provides highest compression, while the Le Gall 5-tap/3-tap reversible wavelet transformation provides lossless compression [6, 7, 29, 9]. The wavelet transformation is applied by filtering each row and column of the pre-processed image tile by a high pass and low pass filter. For keeping the sample rate constant, a down sampling by two (every other value is removed) must be performed, after applying the low pass and high pass wavelet filtering to the pre-processed image data [9]. Figure 2.4 depicts step by step how a one-step wavelet transformation is performed (high- and low-pass filters and a down-sampling by 2 are applied).

2.2.3 Quantization

Even after the wavelet transformation has been applied, the image data is not yet compressed. The wavelet transformation is solely responsible for restructuring the image information in such a way that it is easier to compress the data through quantization. The

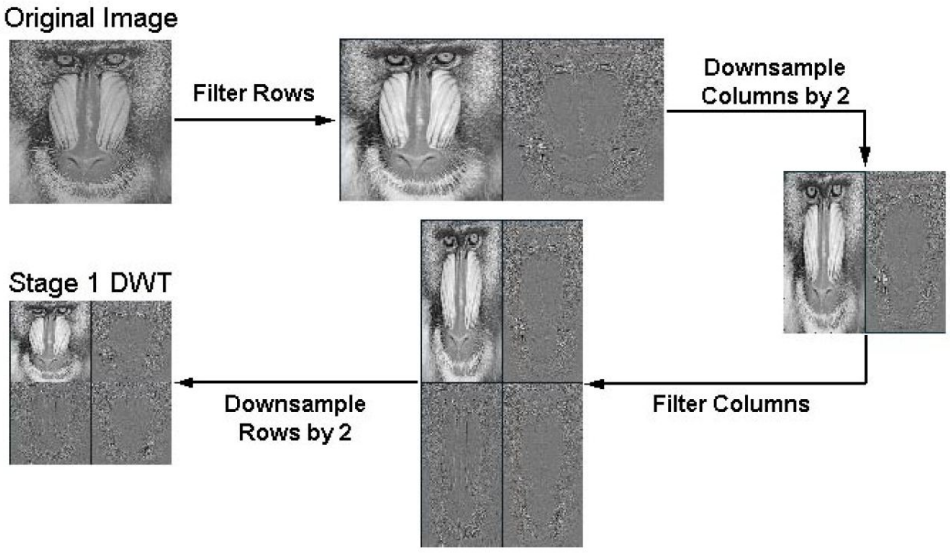


Figure 2.4: The figure above shows the Discrete Wavelet Transformation (DWT) process for the Y component based on the baboon image [7]

quantization process step is responsible for obtaining a trade-off between rate and distortion by quantizing all wavelet coefficients. Therefore, a uniform scalar quantization with dead-zone around the origin is used to compress the image data. This processing step is lossy, unless the quantization step is limited to 1 and the image coefficients are integers, as produced by the 5/3 reversible wavelet transformation [29, 11, 6].

2.2.4 Context Model

Before JPEG2000 arithmetic coder can perform coding, the subbands of each tile are further partitioned into smaller non-overlapping rectangular blocks, so called code-blocks. A typical code-block-size, as stated by Marcellin et al. [6], is 64 $\times$ 64 or 32 $\times$ 32 pixel. Furthermore, as stated by Acharya et al. [32], the compression performance decreases in case a code-block-size less than 16 $\times$ 16 is chosen. The purpose of further dividing the subbands into code-blocks of the same size (except those located at the image borders) is to permit a flexible bitstream organization after encoding the JPEG2000 image through the arithmetic entropy coder [9, 29, 6, 7]. Figure 2.6 shows a sample image partitioning (into subbands and code-blocks).

2.2.5 Arithmetic Entropy Coder

The basic functionality of the arithmetic entropy coder is to remove redundancy in the encoded of the image data. To fulfill this, it assigns short code-words to the more probable events and longer code-words to the less probable ones [11]. In JPEG2000, the coding algorithm encodes each code-block independently without any reference to other blocks

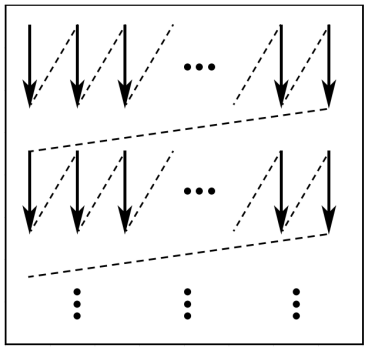


Figure 2.5: Sample scan order within a JPEG2000 code-block [8]

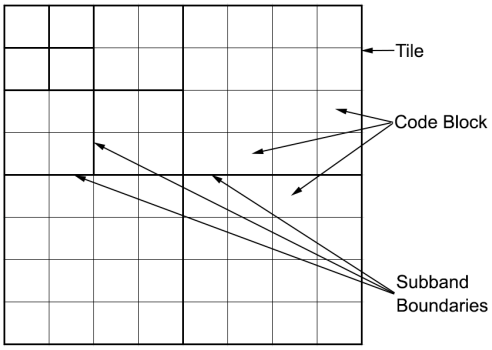


Figure 2.6: Tile partition into subbands and code-blocks [9]

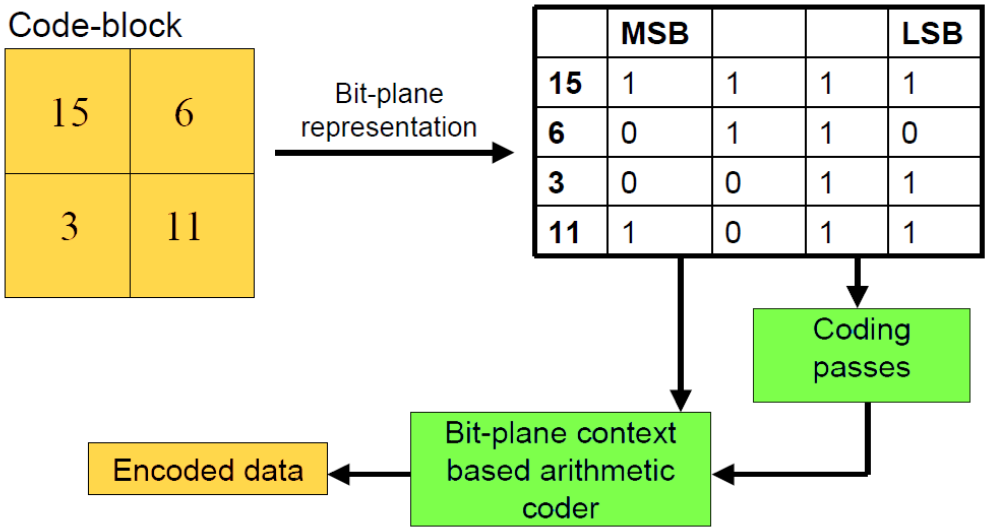


Figure 2.7: Sample JPEG2000 Entropy coding [10]

in the same or any other subband (Figure 2.7 shows a simplicial illustration of how the arithmetic decoder works). Within each subband, the coder scans each bit plane of a code-block in a special order (see Figure 2.5). Starting from the top left, the first four bits of the first column are scanned. The first four bits of the second column are scanned next, until the end of the code-block row is reached. After that, the second four bits of the first column are scanned and so on [29]. Each coefficient bit in the bit-plane is coded in only one of the three coding passes, namely the significance propagation, the magnitude refinement, and the cleanup pass. For each pass, contexts, which are provided to the arithmetic coder, are created. For further details about the arithmetic entropy coder, see the book "JPEG2000 Image Compression Fundamentals, Standards and Practice" by Taubman et al. [9].

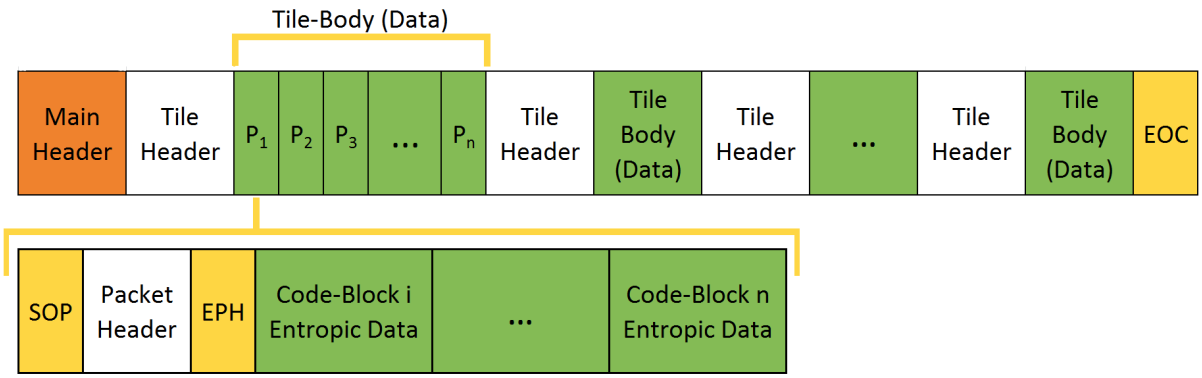


Figure 2.8: JPEG2000 Codestream syntax [11]

2.2.6 Bitstream Ordering

After applying the arithmetic coder to the image data, its output is collected into packets. One packet is generated for each precinct in a tile. A precinct is essentially a container for a number of code-blocks, and is used to facilitate access to a specific area within an image. Furthermore, each packet consists of a header and the encoded image coefficients [29, 6, 7]. After organizing the output of the arithmetic coder into packets, the packets are multiplexed in an ordered manner to form the JPEG2000 code-stream. The JPEG2000 standard defines five ways of ordering the packets (called progressions), as mentioned in Part 1 of the JPEG2000 standard [9, 7].

2.2.7 Codestream Syntax

The final codestream consists of marker segments and the coded image coefficients. These marker segments are used to structure the codestream and consist of two bytes (starting with 0xFF). The marker segments facilitate to determine location of the encoded data corresponding to a given spatial location, resolution, and quality in the image [6]. A more detailed explanation of marker segments is given by Taubman et al., in the book "JPEG2000 Fundamentals" [9]. Figure 2.8 depicts a sample JPEG2000 codestream syntax.

2.2.8 File Format

Any additional data which is related to the image, but is not needed to reconstruct components of the image, is stored in the file format. The optional file format is provided to prevent the proliferation of non-standard proprietary formats, which happened with the original JPEG standard. The file format begins with a unique signature, has a profile indicator, and repeats the width, height, and depth information from the codestream. Optionally, the file format may contain a limited color specification, capture and display resolution, intellectual property rights information, and some additional metadata. The

output of the encoder can either be directly sent to the recipient or stored in a JPEG2000 format compliant form. The JPEG2000 standard defines the file extension .jp2 [6].

2.3 RoI Coding

The JPEG2000 standard exhibits the feature of defining a Region-of-Interest (RoI). This image region (e.g., the facial area of a person, or any other important image region) as defined in Part 1 of the JPEG20000 coding standard can be of arbitrary shape. As the image coefficients belonging to the RoI are shifted by the Max-Shift method, the bits associated with the RoI are placed in higher bit-planes than the bits associated with the background. The shifting procedure is depicted in Figure 2.10. The shifting of image coefficients belonging to the RoI into higher bit-planes leads to the fact that during the embedded coding process, the most significant RoI bit-planes (MSB) are placed in the JPEG2000 bitstream before any background bit-planes of the image. Hence, the RoI will be decoded prior to the rest of the image. Theoretically, if the bitstream is truncated, or the encoding process is terminated before the whole image is fully encoded, the ROI will be of higher fidelity than the rest of the image [9, 29, 33].
As outlined by Skodras et al. [29], the general JPEG2000 Max-Shift method is implemented as follows (focus is on RoI scaling; pre-processing, bitstream ordering, etc. are skipped for simplicity reasons in the following outline):

1. The wavelet transform is calculated
2. In case an RoI is defined, an RoI mask needs to be derived, which indicates all the wavelet coefficients associated with the RoI (Figure 2.10 depicts a sample RoI mask)
3. After defining the RoI mask, the wavelet coefficients are quantized
4. The next step deals with downscaling the background coefficients by a specified scaling factor (Max-Shift scaling factor needs to be high enough to shift the background coefficients below the least significant bitplane (LSB) of the RoI (see Figure 2.9))
5. Finally, all coefficients are entropy encoded, with the most significant bit-plane first (RoI coefficients are located at the beginning of the codestream)

As mentioned above, the Max-Shift method makes it possible to define RoIs of arbitrary shape, without the need of transmitting any additional information concerning the shape or location of the RoIs. Therefore, only one additional marker needs to be added to the JPEG2000 codestream. This marker is the Region-of-Interest marker (RGN), which indicates the up-shift factor by which the background has been downscaled while encoding the

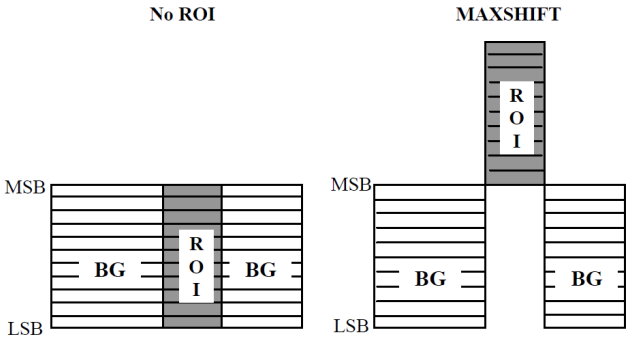


Figure 2.9: Scaling of JPEG2000 RoI coefficients [12]

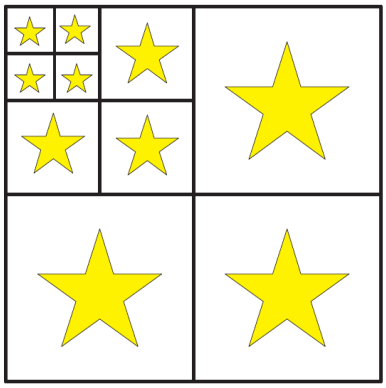


Figure 2.10: Example – JPEG2000 Wavelet Domain RoI mask [7]

image data. The RGN marker can be placed at multiple JPEG2000 bitstream locations, e.g., the main header or in the tile header containing an RoI. For a more detailed explanation of the RGN-marker segment, or any other marker segment, reference is made to Taubman and Marcellin's book "JPEG2000 Image Compression Fundamentals, Standards and Practice" [9].

2.4 JPEG2000 Parts

The term JPEG2000 refers to all parts of this compression standard for still images. Some of the JPEG2000 parts have already been published, such as Part 1 (the core of JPEG2000 coding), which is an international standard for still image compression. Parts 2-6 have been completed or are nearly completed and Parts 8-12 are under development [28]. While the majority will be interested in the core coding system of Part 1, this section describes the reasons for Parts 1-8 (Parts 9-12 will not be covered by this work).

2.4.1 Part 1 - Core Coding System

As its name suggests, Part 1 defines the core functionalities of the JPEG2000 compression standard. This includes the syntax of the JPEG2000 codestream and the necessary steps involved in coding the JPEG2000 image (see Section 2.2 for more details). A number of existing implementations, such as the Java implementation JJ2000, utilize only Part 1, which is sufficient to code JPEG2000 images properly. Part 1 defines a basic file format called JP2, which can be used to store JPEG2000 coded images. Furthermore, Part 1 was developed with the intention of being available without the requirement of any license fees.

Part 1 became an International Standard (ISO/IEC 15444-1) in December 2000 [28].

2.4.2 Part 2 - Extensions

Part 2 defines various extensions to Part 1, which are required by some image compression applications where interoperability is not as important as other requirements. Some of the extensions of Part 2 listed by Marcellin et al. [6] are: the use of another quantization (e.g., Trellis Coded Quantization, which increases rate-distortion performance); allow multiple wavelet transformations; a new file format JPX (used to support multiple layers, animation, extended color spaces, etc.) or the option for additional metadata.
Part 2 became an International Standard (ISO/IEC 15444-2) in November 2001 [28].

2.4.3 Part 3 - Motion JPEG2000

Part 3 defines a file format called MJ2 (or MJP2) for motion sequences of JPEG2000 images. Support for associated audio is also included. However, Part 3 does not include inter-frame coding. Hence, each frame must be encoded and decoded independently. In addition to Part 2, this part is not compatible with Part 1 and consequently a standard compliant Part 1 decoder cannot handle MJ2 files.
Part 3 became an International Standard (ISO/IEC 15444-3) in November 2001 [28].

2.4.4 Part 4 - JPEG2000 Conformance

Part 4 of the JPEG2000 standard defines some test procedures for both encoding and decoding a JPEG2000 image. It provides definitions of a set of decoder compliance classes. Therefore, Part 4 consists of test files including bare codestreams and JP2 files.
Part 4 became an International Standard (ISO/IEC 15444-4) in May 2002 [28].

2.4.5 Part 5 - JPEG2000 Reference Software

Part 5 provides two source code packages, implementing Part 1 of the JPEG2000 standard. One is written in Java (JJ2000[1]) and the other in C (JasPer[2]). The JJ2000 group consisted of Canon France, Ericsson and EPFL, and JasPer was developed by the University of British Columbia. Both are available under open-source type licensing [6].
Part 5 became an International Standard (ISO/IEC 15444-5) in November 2001 [28].

2.4.6 Part 6 - JPEG2000 Compound Image File Format

As its name suggests, Part 6 defines a file format for compound images. Hence, it offers a mechanism by which multiple images can be combined into a single compound image. Furthermore, it defines a new file format called JPM, which is an extension of the JP2 file

[1] http://jj2000.googlecode.com/svn/trunk/
[2] http://www.ece.uvic.ca/ frodo/jasper/

format. It builds upon the same architecture defined in Part 1 and uses many of the same boxes defined in Part 1 (JP2) and Part 2 (JPX). It is used for document imaging (storing multi-page document with many objects per page, for pre-press and fax-like applications). Even though JPM is a JPEG2000 part, it supports many other coding or compression technologies, such as JPEG and JPIG2.
Part 6 became an International Standard (ISO/IEC 15444-6) in April 2003 [28].

2.4.7 Part 8 - JPEG2000 Security

This part of the JPEG2000 standard defines additional measures to secure JPEG2000 compressed images. Part 8 is also known as JPSEC. The JPSEC standard provides a security framework with a wide range of security services, including confidentiality, source authentication, data integrity, conditional access and ownership protection [28, 34, 35]. The JPSEC bitstream is quite similar to the JPEG2000 Part 1 bitstream, except it might be partially encrypted, scrambled or watermarked in order to provide one of the security services mentioned above. To enable a proper decoding of the JPSEC bitstream, two additional marker segments are introduced. These markers are the Security Marker segment (SEC) and the In-Codestream Security Marker segment (INSEC).
The SEC segment is located at the main header of the JPSEC bitstream and it signals some general security parameters used to secure the JPEG2000 bitstream. These parameters are applicable to the whole JPSEC bitstream. To signal the covered areas in terms of both image related parameters (such as tile, resolution, precinct, component and region of interest) and no-image related parameters (such as byte range) in a JPSEC bitstream, the Zone Of Influence (ZOI) has been introduced. However, if each packet of a ZOI is encrypted by a different key, the second marker segment (INSEC) can be used to signal the different keys. In doing so, the decoding performance can be increased, as stated by Sun an Zhishou [36]. The INSEC marker can be placed anywhere in the bitstream because the arithmetic JPEG2000 decoder stops automatically reading the bitstream when encountering any marker segment (two bytes with a value exceeding 0xFF8F) [36]. JPSEC will not be used in this work to detect RoIs in the JPEG2000 bitstream, due to the fact that the objective of this work is to automatically detect, without any additional marker segments, the encrypted image regions. Hence, JPSEC is not applicable, as it offers the functionality to add additional marker segments to the bitstream that indicate the position and type of encryption. Therefore, this paper focuses on detecting encrypted RoIs in a JPEG2000 encoded image, based on Stubhann's RoI bitstream encryption implementation [5], which follows an approach proposed by Hämmerle-Uhl et al. [19]. Part 8 became an International Standard (ISO/IEC 15444-8) in July 2006 [28].

Chapter 3

Multimedia Encryption

Nowadays security and privacy in digital video applications, such as video surveillance systems or digital cinema, are a major concern, especially when it comes to the ease to manipulate, copy, analyze and distribute digital content at negligible cost. Furthermore, these concerns raises, the issues of confidentiality, data integrity, authentication and conditional access control [37]. Hence, to develop privacy-preserving video surveillance systems, the security issue should be taken into account. Therefore, this chapter gives an overview of possible encryption approaches, their evaluation criteria and some approaches on how to encrypt RoIs in a JPEG2000 image. The latter will be of importance to this work, hence this work is based on Stubhann's primary work [5], which deals with an RoI encryption implementation and its evaluation.

3.1 Encryption Approaches

Compared to the encryption of plain text or the encryption of the whole codestream, the encryption of multimedia data is computationally more demanding and complex [5]. There are several reasons for this, including the increased data-volume (images or video-files have in general a larger file-size than plain-text files) and the coding of image data, which is necessary to compress the image data, but increases the encryption complexity. Due to this fact, researchers have developed different approaches to solve the problem of encrypting multimedia data [37, 38, 39]. Basically, researchers have developed methods for encrypting multimedia data that can be applied at three different image-compression-stages. These stages are: in the image-domain prior to coding; in the transform-domain during coding; and in the codestream-domain after coding [37]. All these approaches are more thoroughly discussed hereafter.

3.1.1 Pre-Compression / Image-Domain Encryption

The first encryption approach performs image data encryption prior to encoding the image. Compared to the other encryption approaches, Pre-Compresssion is very simple. This is based on the fact that encryption can be performed on unmodified (not encoded) image data, hence it is independent from any encoding processes.
However, this simple and fast approach has two major disadvantages. The first one is that it significantly alters the image data statistics, hence making the ensuing compression less efficient [37]) The other disadvantage is that the amount of data which must be encrypted is higher at this stage than after or while coding [5].

3.1.2 In-Compression / Transform-Domain Encryption

The second encryption approach applies multimedia data encryption while encoding the JPEG2000 image. As stated by Mao et al. [40], the encoding process offers multiple stages for inserting the encryption method. Some of these stages are Wavelet Transform, after or while Quantization and the Arithmetic Entropy Coder (for further details about these stages see Section 2.2) [40, 5]. One benefit of this approach is that, thanks to the frequency analysis property of the wavelet transform, the strength of the encryption can be controlled by restricting the encryption to some frequencies [37]. Nevertheless, manipulating the encoding process poses the risk of losing JPEG2000 format-compliance [5]. However, as shown by Dufaux et al. [37], losing format-compliance can be prevented by designing the encryption method according to the JPEG2000 standard. Another major drawback of this method is based on the fact that no standard compliant encryption cipher (e.g., Advanced Encryption Standard AES) can be applied at this stage of encryption. This is due to the fact that the encryption cipher needs to be adjusted in a way that it does not compromise the JPEG2000 coding procedure. Hence, image data security cannot be guaranteed. Furthermore, encrypting the image while encoding the image data can result in additional overhead, due to altered image data statistics, which make the ensuing compression less efficient [5].

3.1.3 Post-Compression / Codestream-Domain Encryption

Finally, the third encryption approach applies image encryption after encoding. More specifically, this approach encrypts the coded JPEG2000 bitstream directly [37]. This approach poses the advantage of being able to use any standardized cryptographic cipher (e.g., AES) to encrypt the JPEG2000 bitstream. Therefore, image data security can be guaranteed if the encryption method has been applied correctly [41]. However, in addition

to encrypting In-Compression, this approach poses the risk of losing format-compliance when encrypting the image data without taking the JPEG2000-marker segments into account [40]. Thus, one of the drawbacks of this approach is that the codestream needs to be parsed in order to identify which parts correspond to the regions to be encrypted, hence entailing a larger computational complexity [37]. Another drawback this approach has in common with the In-Compression approach is the additional overhead caused by storing the encryption information (encrypted JPEG2000 packets, encryption-counter, etc.) [5].

3.2 Evaluating the Encryption Methods

As the approaches proposed for JPEG2000 encryption differ significantly in their field of application (see Section 3.1), their level of security, the functionalities they provide and their computational demands, a systematic evaluation needs to take place when comparing the various encryption techniques [4]. Therefore, this section covers criteria by which JPEG2000 encryption techniques can be evaluated and classified.

3.2.1 Format Compliance

Each multimedia compression standard defines its own codestream syntax, which includes the placement of marker segments, the byte value of these marker segments and header structures. This codestream syntax is sometimes called meta-information, which is required by a standard compliant decoder to decode the codestream properly. The objective of format-compliant encryption is therefore to preserve these selected parts of the codestream in a way that the encrypted data is still format-compliant. If format-compliance is desired, the classical naïve cryptographic approach (the whole codestream is encrypted) cannot be employed as the meta-information would not be preserved. Hence a standard compliant decorder would no longer be able to decode the encrypted bitstream, except if the decryption of the encrypted bitstream has been performed beforehand [4]. Therefore, this evaluation criterion is used to evaluate whether the proposed encryption method fulfills the format specified by the multimedia standard in use (e.g., JPEG2000).

3.2.2 Overhead

In this context, overhead is defined as the increase of data after encrypting the image data (given in bytes). There are several reasons for this, including altering the image data statistics, which makes the ensuing compression less efficient. Another reason is the additional data, which is required to decrypt the encrypted image parts (e.g., cipher-key, position and length of encrypted bitstream section, encryption-counter, encryption-

cipher-method, etc.). Because of this, minimizing the amount of additional data required to decrypt the encrypted image parts is a crucial criterion for evaluating an encryption technique [5].

3.2.3 Computational Demand

As encrypting multimedia data evokes an additional processing step while encoding the multimedia data, this criterion will be outlined shortly in this subsection. The additional processing step can either result in a longer computing time or in an increase of computational power required to complete the coding. Basically, the format-compliant encryption of JPEG2000 images is computationally more demanding than a naïve encryption of the whole codestream, as the structure of the JPEG2000 codestream needs to be preserved. The employment of SOP and EPH markers (both are optional, but used in this work to minimize overhead), which is used to indicate the position of the JPEG2000 packet body data, reduces the cost of parsing the JPEG2000 codestream. Therefore, when using these optional markers, the overhead and the computational demand can be limited to a certain degree, as not every encrypted bitstream section needs to be stored [42].

3.2.4 Security

The security aspect, as the name suggests, is very important when it comes to evaluating different encryption approaches. An encryption method which can be broken with negligible effort is of limited use when it comes to encrypting multimedia data. Because of this, the encryption approaches proposed by the authors of In-Compression encryption methods (see Section 3.1.2) should be evaluated thoroughly, as these approaches need to use ciphers, which are not as thoroughly tested or evaluated for security issues as the standardized cipher AES (Advanced Encryption Standard), for instance. Another security aspect which needs to be evaluated is based on the visual recognition of the encrypted parts of an image by an un-authorized observer. Thus, if the image content is still recognizable after encrypting the image, the whole encryption processes is useless [5].
Furthermore, as pointed out by Engel et al. [4], it is not sufficient to solely encrypt the packet body of a JPEG2000 image, as the packet headers contain crucial (even visual) information about the source image as well, which can be used to figure out what the encrypted part of the image is all about. Therefore, this aspect needs to be taken into account while evaluating an encryption approach.

3.2.5 Transcodability

Another evaluation criterion for multimedia encryption methods is the transcodability. Transcodability is the ability to convert the coded multimedia data into another format, which might be necessary for some applications. Therefore, when encryption has been applied to the codestream, it is important to note whether it is possible to convert the codestream directly into another form, or if some intermediate processing step (decrypting the codestream – convert to other format – encrypt codestream again [34]) must be executed beforehand. A successful and fast Transcodability is therefore strongly linked to format-compliance, which, when preserved, might make the intermediate step unnecessary [5].

3.2.6 Image Quality

As pointed out by Engel et al. [4] and Köckerbauer et al [43], the peak-signal-to-noise-ratio (PSNR) is no longer an optimal choice for assessing image quality. However, the PSNR is widely used because it is unrivaled in speed and ease of use [19]. The PSNR is calculated as follows [44]:

$$PSNR = 10 \log_{10}(\frac{M^2}{MSE}) \tag{3.1}$$

where M is the maximum possible pixel value of the image, and MSE the mean squared error defined as

$$MSE = \frac{1}{WH} \cdot \sum_{i=1}^{W} \sum_{j=i}^{H} (I(i,j) - O(i,j))^2$$

The MSE is applied to two images. The original image O and impaired image I, both with size $W \times H$ (width and height). Due to the fact that the PSNR is no longer an optimal choice for assessing image quality, the authors Engel et al. [4] and Köckerbauer et al [43] proposed to use the state-of-the-art image quality measure SSIM (structural-similarity-index-measure) for getting an adequate image quality measure, which comes closer to the quality assessment of the human visual system. The SSIM ranges, with increasing similarity, ranges from 0 to 1, where 0 indicates that the images are very highly dissimilar and 1 indicates that the two images are identical. The SSIM between two images is calculated as follows (as described by Wang et al [45]):

$$SSIM(I,O) = \frac{(2\mu_I\mu_O + c_1)(2\sigma_{IO} + c_2)}{(\mu_I^2 + \mu_O^2 + c_1)(\sigma_I^2 + \sigma_O^2 + c_2)}, \tag{3.2}$$

where μ_I is the average pixel value of image I, σ_I^2 is the variance of pixel values of image I and σ_{IO} is the covariance of I and O. The variables $c_1 = (k_1 M)^2$ and $c_2 = (k_2 M)^2$, with $k_1 = 0.01$ and $k_2 = 0.03$, are used to stabilize the division.

Furthermore, as pointed out by Engel et al. [4], there is a measure specifically for the security evaluation of encrypted images that separates luminance and edge information into a *luminance similarity score* (LSS) and an *edge similarity score* (ESS). These measures are based on the work proposed by Mao and Wu [40] that tried to find a measure matching the optical characteristics of a human eye. As stated by Mao and Wu [40], the human eyes can extract coarse visual information in images and videos in spite of a small amount of noise and geometric distortion. Hence, they proposed these two metrics of evaluating image quality, which were better than with the PSNR.

ESS is a score measuring the degree of resemblance of the edge and contour information between two images on a block basis. It delivers a value ranging from 0 to 1, where 0 indicates that the edge information of the two images is highly dissimilar and 1 indicates a match between the edges in the two images. Denoting e_{1i} and e_{2i} as the edge direction indices for the i-th block in two images, respectively, the edge similarity score (ESS) for a total of N image blocks is computed as follows [40]:

$$ESS \;\widehat{=}\; \frac{\sum_{i=1}^{N} w(e_{1i}, e_{2i})}{\sum_{i=1}^{N} c(e_{1i}, e_{2i})} \tag{3.3}$$

$$w(e_{1i}, e_{2i}) \;\widehat{=}\; \begin{cases} 0 & \text{if } e_1 = 0 \text{ or } e_2 = 0, \\ |\cos(\phi(e_1) - \phi(e_2))| & \text{otherwise,} \end{cases}$$

where $\phi(e)$ is the representative edge angle for an index e, and $c(e_1, e_2)$ an indicator function defined as

$$c(e_{1i}, e_{2i}) \;\widehat{=}\; \begin{cases} 0 & \text{if } e_1 = 0 \text{ or } e_2 = 0, \\ 1 & \text{otherwise,} \end{cases}$$

The LSS determines the similarity of the luminance components and has a variable range depending on the color depth and the chosen parameters. The Luminance similarity score is calculated on block basis. Therefore, the two images are divided into non-overlapping blocks in the same way. Then the average luminance values of the i-th block from both images, y_{1i} and y_{2i}, are calculated. Therefore, Mao et al. [40] have defined the LSS as follows:

$$LSS \;\widehat{=}\; \frac{1}{N} \sum_{i=1}^{N} f(y_{1i}, y_{2i}), \tag{3.4}$$

Here, the function $f(y_{1i}, y_{2i})$ for each pair of average luminance values is defined as

$$f(y_{1i}, y_{2i}) \;\widehat{=}\; \begin{cases} 1 & \text{if } |x_i - x_2| < \frac{\beta}{2}, \\ -\alpha \ \text{round}(\frac{|x_i - x_2|}{\beta}) & \text{otherwise,} \end{cases}$$

where the parameters α and β control the sensitivity of the score. As stated by Mao et al. [40], image comparison may be corrupted by noise during transmission or minor pixel perturbation. Therefore, they proposed the scaling factor α and the quantization parameter β to improve resistance to noise and minor perturbation. A negative LSS value indicates substantial dissimilarity in the luminance between the two images. For example, as stated by Köckerbauer et al. [43], this yields for an 8 bit per pixel image ($\alpha = 0.1$ and $\beta = 3$) an LSS range of -8.5 to 1, where -8.5 indicates the worst image quality and 1 indicates that the images are identical.

3.3 JPEG2000 RoI Encryption

This section gives an overview of JPEG2000 format-compliant RoI encryption approaches applied at codestream level. This section is divided into three main parts. The first part describes three different approaches used to detect the data packets belonging to an RoI. The second part handles the JPEG2000 packet body encryption. Finally, the third part focuses on encrypting the packet headers, which have been shown by Engel et al. [4] to contain crucial information about the source image. However, as mentioned in the Section 1.1, this work is based on Stubhann's implementation of a bitstream RoI encryption, which follows an approach proposed by Hämmerle-Uhl et al. [19].

3.3.1 Detect RoIs based on Max-Shift

The Max-Shift method is defined as the standard RoI coding method in the JPEG2000 Part 1 compression standard for still images [9]. As mentioned in Section 2.3, an RoI can be of arbitrary shape. The image coefficients belonging to the RoI are shifted by the Max-Shift method, so that the bits associated with the RoI are placed in higher bit-planes than the bits associated with the background. For decoding the JPEG2000 image, the scaling value is extracted from the JPEG2000 RGN-segment, which is used while decoding the JPEG2000 image to up-scale the image coefficients smaller than the scaling value, as these coefficients belong to the background. The RoI scaling procedure leads to the fact if *quality progression bitstream ordering* has been applied to the JPEG2000 codestream that all data packets belonging to an RoI are aligned at the beginning of the JPEG2000 codestream. Because of this, it is sufficient to store only the length of the encrypted

(a) 4x4 **(b)** 8x8 **(c)** 16x16 **(d)** 32x32 **(e)** 64x64

Figure 3.1: Example RoI Max-Shift encryption with varying code-block-sizes (surveillance image cam1_1 from the SCFace image database [46], - image size = 75x100 pixel, RoI = 36x72 pixel and Wavelet Decomposition Level = 0)

codestream. Furthermore, the decoder is able to extract the encrypted JPEG2000 data packets belonging to an RoI by parsing the codestream (packet body data is embraced by Start of Packet (SOP) and End of Packet (EOP) marker segments) [39]. Figure 3.1 depicts some sample results caused by varying code-block-sizes.

However, the major drawback of this approach is that the bitstream needs to be encoded in *quality progression bitstream order* and the SOP and EOP marker segments need to be used while encoding the image [9]. If any other JPEG2000 bitstream ordering has been used or the mentioned marker segments have not been embedded while encoding the image, the overhead caused by storing the encryption specification will be much higher. Hence, every start- and end-point of an RoI data packet needs to be stored.

3.3.2 Detect RoIs by Codeblocks

The second RoI detection approach relies on the JPEG2000 code-block image structure. Hence, everything required to detect the RoI accurately is the code-block-size and the Wavelet decomposition level used while encoding the JPEG2000 image. Furthermore, it should be noted that the initial encoding code-block-size is defined for the first resolution level (lowest resolution) and increases by the factor 2 for each performed Wavelet decomposition (resolution level). Prior to encrypting any code-blocks, it needs to be determined which code-blocks in which resolution levels represent the RoI's spatial extent. However, to determine the code-blocks belonging to an RoI accurately, the bitstream ordering used while encoding the JPEG2000 image needs to be taken into account. The bitstream ordering influences the alignment of the code-block within the JPEG2000 codestream. After the code blocks belong to the RoI are detected, they can be encrypted. The decryption of the encrypted JPEG2000 code-blocks is performed in the same way, provided that the same information (initial code-block-size, used Wavelet-levels) required to encrypt the code-blocks is available. Furthermore, it should be noted that this encryption approach features the possibility to scale the encryption quality, as each code-block of each resolution and subband can be accessed separately [39].

3.3.3 Detect RoIs by Tiles

Finally, the third RoI detection approach relies on the fact that each JPEG2000 image is further divided into non-overlapping rectangular tiles. Similar to the codeblock RoI detection method, this method utilizes the fact that tiles can be used to access certain image regions. Prior to detecting the tiles belonging to an RoI, it should be noted that inside the JPEG2000 codestream the tiles are always ordered in the same way. Due to the fact that the source image is partitioned into non-overlapping tiles, these tiles need to be aligned somehow in the JPEG2000 codestream. Therefore, the tiles are ordered starting by the top left corner of the source image and continuing until the bottom right is reached. However, very small tile sizes, which might be required to accurately cover the RoI, negatively influence the compression efficiency. The encryption of the RoIs, as stated by Hämmerle-Uhl et al. [19], is performed quite similarly to the RoI detection method outlined above. At first, the JPEG2000 codestream is parsed until a Tile Header (TH) is found. Afterwards, it is checked whether the tile belongs to the RoI. Hence, if the tile belongs to an RoI, all packets belonging to the tile will be encrypted. However, while encrypting the packets, attention needs to be paid to the JPEG2000-marker segments, which should not be modified due to the fact that otherwise, JPEG2000 format-compliance may no longer be given. Due to the tile marker segments, parsing the codestream is less complex and therefore less computationally demanding than the Codeblock method (see Subsection 3.3.2). However, the major disadvantage of this method is that each tile-header adds additional overhead to the codestream. This leads to low compression efficiency, for small tile size in particular [39].

3.3.4 Packet-Body Encryption

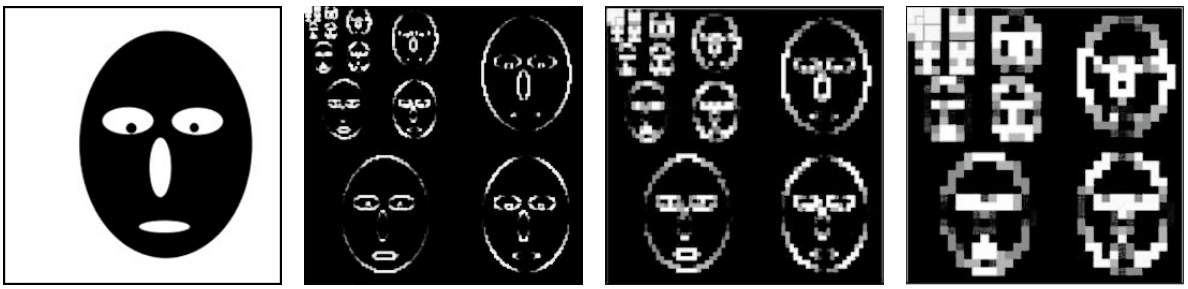
As stated by Stütz and Uhl [41], all format-compliant codestream encryption methods proposed thus far apply the RoI encryption on the JPEG2000 packet bodies containing the compressed image data. The packet bodies are responsible for storing the encoded image coefficients. All the other data stored within the JPEG2000 codestream are used to store additional information, e.g., compression parameters, data structure, etc., which is used to decode the image. However, the JPEG2000 codestream syntax imposes certain requirements on the packet data format, which cannot be guaranteed by any standard encryption method (no additional marker segments, no change in packet size, and no packet is allowed to end with the byte value 0xFF). Therefore, the iterative encryption approach proposed by Wu and Deng [47] is capable of encrypting 100 % of the packet body data while not producing any additional markers or harming format-compliance.

The basic encryption algorithm proposed by Wu and Deng [47] looks as follows:

1. Encrypt the packet body
2. Check if it contains a two byte sequence in excess of 0xFF8F
   If yes, go to 1 and re-encrypt the encrypted packet body
3. Check if it ends with 0xFF
   If yes, go to 1 and re-encrypt the encrypted packet body
4. Output the format-compliant codestream containing the encrypted packet body data

3.3.5 Packet-Header Encryption

This subsection discusses format-compliant packet header encryption. As shown by Engel et al. [4], the packet-header contains information of the source image. However, even if the encoded image coefficients are stored in an encrypted form in the packet-bodies, it is possible to extract information from the packet headers. Particularly for high-resolution images or small codeblock-sizes (see Figure 3.2), the risk of not encrypting the leading zero bitplanes (LZB) in the packet headers can harm content security/confidentiality.
When applying the packet-header encryption proposed by Engel et al. [4], no influence on compression performance is observed. Furthermore, if the visual information contained in LZB information is effectively encrypted by the proposed encryption approach, content security/confidentiality can be achieved. However, as stated by Engel et al. [4], even if packet body based encryption and format-compliant header encryption are combined, security under *Indistinguishability under chosen-plaintext attack* can still not be achieved as the packet borders are preserved. The proposed format-compliant header encryption method invokes only small computational overhead, due to the fact that the packet header data accounts only for a small fraction of the actual codestream. For a more detailed explanation of the packet-header encryption, reference is made to Engel et al. [4]



(a) Original, image 2160×2160 pixel [4]

(b) Codeblock-size 16×16 [4]

(c) Codeblock-size 32×32 [4]

(d) Codeblock-size 64×64 [4]

Figure 3.2: The figures above show the effect of extracting the visual information of JPEG2000 packet headers from its leading zero bitplanes (LZB). The example is based on a high resolution source image (see Figure 3.2a). As seen in the figures, the code-block-size used while encoding the JPEG2000 image contributes to the visual information extractable from the packet-headers (see Figure 3.2b, 3.2c, 3.2d) [4]

Chapter 4

Data Embedding Techniques

Embedding data into a media file can basically be performed in two closely related ways, namely steganography and watermarking. These embedding techniques have a great deal of overlap and share many technical approaches. Information hiding, which is a general term including the subdisciplines watermarking and steganography, have recently become more and more important in a number of different application areas (e.g., Digital audio, video, and pictures) [48, 49, 13].
This chapter gives a basic understanding of steganography, which is used for secret communication and watermarking. Watermarking is used for content protection, copyright management, content authentication and tamper detection. Furthermore, this chapter describes some data embedding techniques targeting the JPEG2000 standard.

4.1 Watermarking

As mentioned in the introduction, watermarking is a subdiscipline of information hiding which is used to hide proprietary information in digital media such as photographs, digital music, or digital video [13]. With the increased distribution of digital media through the internet, the unauthorized use of these has increased dramatically and causes billions of euros of loss every year for the media industry. Therefore, digital watermarking has been proposed as a promising technique for information assurance in digital media [50].
However, applying watermarking techniques to the digital media can cause some permanent distortion. Hence, the original media file may not be able to be reversed exactly, even after the hidden data have been extracted from the media file. Because of this, digital watermarking techniques can primarily be referred to lossy data hiding methods. Furthermore, as stated by Cox et al. [13], most of the watermarking algorithms reported in the literature are lossy. This section provides a short description about the watermarking process, the main requirements by which watermarking algorithms can be sorted and some of the watermarking-applications.

4.1.1 Digital Watermarking Process

The process of embedding the watermark (e.g., copyright logo, meta data, etc.) in a media object (e.g., image, video, audio, or any other digital content) is called watermarking. Therefore, a watermark can be considered as a kind of signature that reveals the owner of the multimedia object.

A watermarking system is usually split into two main processing steps, which are embedding and detecting. The embedding of a visible or invisible watermark in a multimedia object is performed by a watermarking algorithm, which decides about the embedding location within the multimedia object. Once the watermark is embedded, it can be attacked in several ways. This is based on the fact that the multimedia object can be processed digitally with ease and at negligible cost. However, even if a person makes any unintentional modification (e.g., filtering, resampling, compressing, etc.), this is considered an attack. Hence, the watermark should be very robust against any unintentional modification a person may execute when using the multimedia object. The watermark can be extracted using the secret key for embedding the watermark. If the extracted watermark resembles the embedded watermark, it can be assumed that no attack has taken place. To check whether an attack against the watermark has taken place, either the original-watermark can be used to extract and compare the embedded-watermark (non-blind watermarking) or a correlation measure can be used to detect the strength of the watermark signal from the extracted watermarked multimedia object (blind watermarking). Therefore, a statistical correlation test is used to determine the existence of the watermark, solely by knowing the key used during the embedding process [13].

Figure 4.1 shows the basic steps of a digital watermarking process.

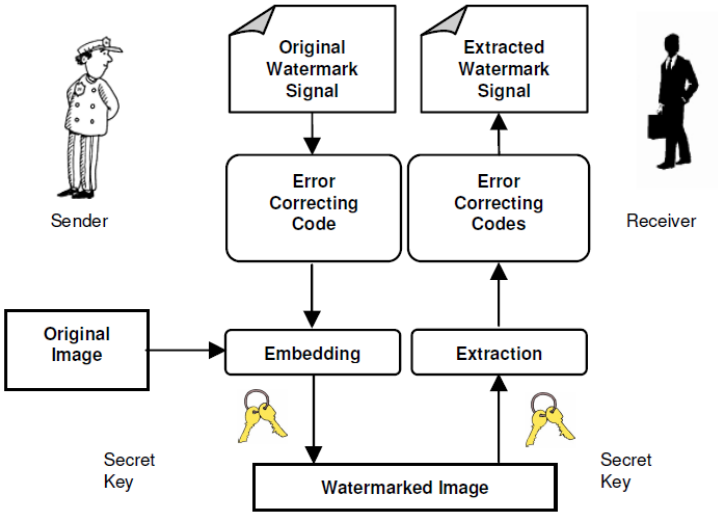


Figure 4.1: Digital Watermarking Process [13]

4.1.2 Requirements of Digital Watermarking

Following are some properties of the watermarking systems. Based on these properties, the overall efficiency of a watermarking technique can be judged [13].

1. **Transparency or Fidelity:** This is basically the perceptual similarity between the original and the watermarked version of the digital media object [51]. Therefore, one evaluation criterion of the watermarking algorithm is that the quality of the watermarked digital multimedia object should not be degraded. If visible distortions are introduced, the commercial success of the watermarking approach can be reduced.

2. **Robustness:** This is the ability to detect the watermark after common signal processing operations have been applied to the digital media object [51]. There are many unintentional watermark attacks (e.g., filtering, scaling, cropping, compressing, etc.) which are not intended to destroy or attack the watermark. Because of this, watermarks should be invariant to a variety of such attacks (unintentional and international), which makes robustness the most important requirement of a watermark.

3. **Capacity or Data Payload:** This is the number of bits a watermarking algorithm is capable of embedding into a digital media object [51]. Therefore, this watermarking property describes the maximum amount of data that can be embedded as a watermark into the image. Furthermore, it ensures a successful detection and extraction of the watermark. The watermark capacity varies from application to application, but, as stated by Potdar et al. [13], it should at least be able to carry enough information to represent the uniqueness of the multimedia object.

4.1.3 Watermarking Applications

Watermarking can be used in a variety of application areas. Some of the main applications, as stated by Vidysagar et al. [13], are:

- **Copyright Protection:** This is by far the most prominent digital watermark application these days. Considering the fact that more and more multimedia objects have been illegally redistributed over untrusted networks such as the internet or any peer-to-peer (P2P) network, a solution needs to be found to solve this issue. Due to this reason, copyright protection tries to counteract the illegal distribution of multimedia objects by embedding watermarks within the digital object. One example of how watermarking can be used to counteract the illegal distributions of multimedia objects is by applying content-aware networks (p2p) that incorporate watermarking technologies to report or filter out copyrighted material from such networks.

- **Authentication:** In some applications, it is required to verify the ownership of the content. One way of achieving this is by embedding a watermark and providing the owner with a private key which gives him access to the message. Hence, the authentication is achieved by comparing the watermark embedded in the multimedia object and the pre-stored watermark (the original watermark) [52].

- **Content Archiving:** A digital watermark can be used to embed a digital object identifier or a serial number into the multimedia object by which the archiving of digital contents (e.g., images, audio- or video-files) is supported. Furthermore, watermarks can be used to support the automated classifying and organizing of digital contents. Commonly, digital contents are identified by their file name. This is a very fragile technique compared to to embedding a digital object identifier within the multimedia object itself, as file names can be changed very easily.

- **Meta-data Insertion / Content-Labeling:** Watermarks can be used to provide additional information about the multimedia object. This process of embedding information into the multimedia object is called Content-Labeling or Meta-data insertion (additional information describes the data further). For example, images can be labeled with its (information describing the image) content, which consequently can be used by search engines to get additional information about the image. Other examples of content-labeling might be adding the lyrics or the name of the singer to an audio file, a journalist using photographs of an incident to insert the cover story of the respective news, or even medical x-rays being used to store patient records.

- **Broadcast Monitoring:** As the name suggests, broadcast monitoring is a technique used to verify whether the content (e.g., advertisement) that was supposed to be broadcasted (on TV or radio) has really been broadcasted or not. To fulfill this requirement, watermarking can be used for broadcast monitoring. This feature helps the advertising companies see whether their advertisements appeared at the right time and for the right duration. Hence, this application has the potential to be a financially successful watermarking application.

- **Tamper Detection:** Fragile watermarks can be used to detect tampering in multimedia objects. If the fragile watermark is degraded or destroyed in any way, it indicates the presence of tampering. Hence, this is a sign of no longer trusting the digital content. This feature is very important for applications such as medical imagery and satellite imagery, as these applications rely on highly sensitive data. Another field of application where tamper detection might be useful is the court, where digital

watermarks could be used as a forensic tool to prove whether an image has been tampered with or not.

- **Digital Fingerprinting:** Digital Fingerprinting is a technique used to detect the owner of the digital content by embedding a unique identifier within the multimedia object. Therefore, similar to human fingerprints, the digital fingerprints are unique to the owner of the digital content. A single multimedia object can have different fingerprints as they might belong to different users.

4.2 Steganography

Steganography refers to the science of invisible communication. The word steganography is derived from the Greek words *stegos* meaning *cover* and *grafia* meaning *writing*, defining it as *covered writing*. The goal of steganography is therefore to prevent the detection of the message itself by an observer (person not privileged to see the message), unlike cryptography, where the goal is to secure communication from an eavesdropper. To accomplish this task, steganography hides information in existing parts of the multimedia object (e.g., uses the LSB to hide the secret message), with the intention of hiding the existence of the communicated message [53, 14, 54].
The main difference between steganography and watermarking is the absence of an active attack. This is based on the fact that in steganography the message is transmitted hidden and nobody knows about its presence, except for the communicating parties. However, in conjunction to this work, the host signal (e.g., image, audio file, etc.) cannot be chosen freely, as the encryption specification needs to be stored within the JPEG2000 codestream. Hence, a potential attacker knows about the presence of a hidden message and can either destroy it (by rearranging image coefficients, for example) or try to detect and read it. Because of this, the concepts of steganography cannot be used by this work.
In watermarking, which can be used for copyright protection or authentication (for more applications see 4.1.3), a potential attacker knows about the presence of a watermark. This leads to the fact that watermarking applications are constantly exposed to attacks that would remove, invalidate or forge the watermark. In steganography, there is no such active attack, as there is no value associated with the act of removing the information hidden in the content of the multimedia object. Although steganography is not exposed to constant attack as watermarking is, it should be robust against accidental distortions (e.g., cropping, resampling, filtering, etc.) [14].
This section provides a short description about the steganography concept and the main requirements by which a steganography algorithm can be evaluated.

4.2.1 Steganography Concepts

Based on the paper proposed by Kharrazi et al. [14], this section gives an overview of the underling concepts and definitions used in the field of steganography. Therefore, this section starts by describing the underling concepts of steganography based on the *prisoners problem* [55], where Alice and Bob are two inmates who wish to communicate in order to hatch an escape plan. Although all the messages exchanged between Alice and Bob are examined by the warden called Wendy, Alice wants to send a secret message to Bob (see Figure 4.2). In order to achieve this, Alice embeds m (secret/hidden message) into a cover-object c, and obtains a stego-object s, which is sent to Bob.
This leads to the following definitions, used in steganography:

- **Cover-object:** This is the object used to carry the message. Many different object-types can be used to embed the secret message. Some examples are images, audio, and video, as well as file structures and html pages, etc.
- **Stego-object:** After embedding the secret/hidden message m into the cover object, the resulting object is called *stego-object*, which carries the message.
- **Steganalysis:** This refers to the statistical tests or techniques that help Wendy distinguish between cover-objects and stego-objects. When making this distinction, Wendy does not know the secret key Alice and Bob may be sharing and the warden cannot be sure about the specific algorithm they might be using.

However, it is generally considered that the algorithm in use between Alice and Bob is publicly known, but the key used by the algorithm is kept as a secret between the two parties. This security-assumption is also known, as the Kerchoff's principle in the field of cryptography, which states that while the algorithm can be publicly known, as long as the key is secret, nobody will be able to decrypt the encrypted message. Wendy has no knowledge about the secret key the communication partners Alice and Bob share; however she might be aware of the algorithm they use for embedding the hidden messages.
Wendy, the warden, has the possibility to either passively or actively examine the messages exchanged between Alice and Bob. The difference between a passive and an active examination lies in the fact that, in a passive examination, the warden simply tries to detect any hidden message by applying steganalysis. In an active examination, Wendy can alter messages deliberately, even though she does not see any traces of a hidden message. This is a precaution applied to destroy any potential hidden message exchanged between Alice and Bob. However, the amount of changes made to the multimedia object is limited, as the multimedia object should not show significant visual quality changes.

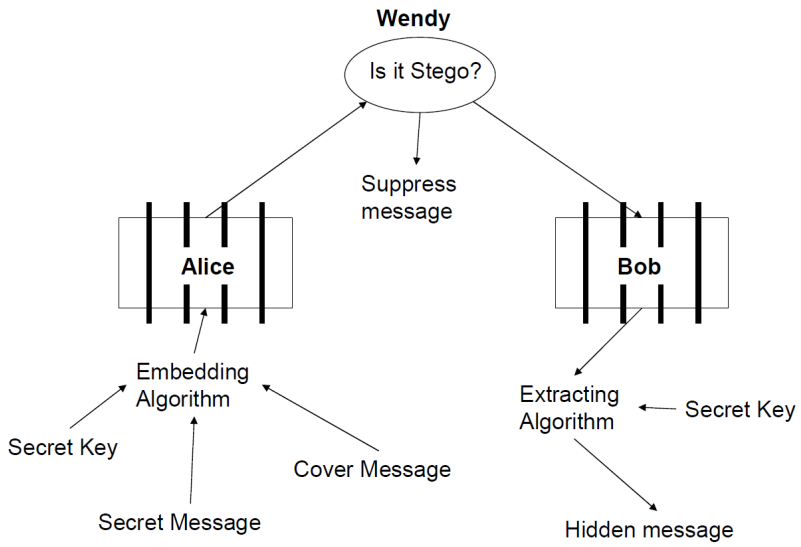


Figure 4.2: General model for steganography [14]

4.2.2 Requirements of Digital Steganography

To evaluate different stenographic algorithms, the authors Morgek et al. [56] and Wei-Ming [54] have proposed some requirements by whitch a steganography algorithm can be evaluated and compared with others.

- **Imperceptibility:** The invisibility of a stenographic message embedded into a multimedia object is the foremost requirement of a stenographic algorithm. Therefore, the information needs to be embedded in a sophisticated way to avoid degrading the perceptual quality of the multimedia object (user cannot see or hear the existence of a hidden message). To achieve this, many information hiding techniques make use of certain human perceptual models in the embedding process. If this requirement is not fulfilled or destroyed by any un-/ or intentional attack, the algorithm is compromised.

- **Capacity:** This property describes the maximum amount of data that can be embedded into the image to ensure invisibility of the hidden message. Unlike watermarking, which needs to embed only a small amount of data (e.g., copyright-, authentication-, archiving-information, etc.) into a multimedia object, steganography aims at hiding a whole communication, meaning that it requires a sufficient embedding capacity.

- **Robustness against manipulation:** A steganography algorithm should be resistant to multimedia object manipulation (e.g., rotating, filtering, resampling, etc.), which can be performed while processing the object. Depending on the manner in which the message is embedded, these manipulations may destroy the hidden message. Because of this, it is preferable for stenographic algorithms to be robust against

either malicious or unintentional modifications of the multimedia object.

- **Robustness against statistical attacks:** By using statistical test (i.e. steganalysis), an attacker tries to detect a hidden message in a multimedia object. Many simple stenographic algorithms leave a so called "signature" when embedding information into the multimedia data. Hence, the signature can be detected with ease by applying statistical tests to the multimedia object. Therefore, the robustness against statistical attacks is another requirement a stenographic algorithm should exhibit in order to be imperceptible and not detectable using statistical tests.
- **Independent of file format:** Constantly using the same file format for inserting hidden messages might be suspicious. Therefore, a powerful stenographic algorithm must cope with multiple file formats, which means the algorithm needs to be able to embed information in multiple file types. Furthermore, the ability to use different file formats solves the problem of converting the multimedia object into the suitable file format or of find a suitable object at the right moment.
- **Unsuspicious files:** However, there is another requirement a steganography algorithm needs to fulfill in order to be difficult to attack and/or not to look suspicious. This requirement includes all the multimedia-object-characteristics that might have changed due to embedding the hidden message. For instance, an abnormal file size might be a sign of an embedded message, which can result in further investigation by an attacker.

4.3 Embedding Binary Data into the JPEG2000 Codestream

In addition to watermarking (see Section 4.1) and steganography (see Section 4.2), additional approaches for embedding RoI information into the JPEG2000 bitstream exist. Some of them are discussed in this section, with respect to the following aspects:

1. **Format compliance:** the strict fulfillment of all syntactical and semantically requirements imposed by the multimedia standard (e.g., JPEG2000) [57, 58]
2. **Losslessness:** the exact preservation of all (visible) picture data [58]
3. **Length-preservation:** the guarantee that the picture's file size does not change (suitable for length-preserving encryption methods) [58]

The following JPEG2000 binary data embedding methods are described with regard to the aforementioned aspects. All RoI embedding methods proposed in this section are summarized in Table 4.1.

4.3.1 Embed Data into the JPEG2000 COM-Segment

JPEG2000 Comment marker segment (COM), which is optional in the JPEG2000 standard, is used to embed unstructured data (e.g., text) into JPEG2000 codestream. To be more precise, the COM-segment is capable of storing up to 65530 payload bytes, plus its marker (2 bytes for COM: 0xFF64), length of marker segment (2 bytes for Lcom: ranging from 4 to 65534 bytes) and the registration values (2 bytes for Rcom: indicates the used data type), totaling 65534 bytes, in the JPEG2000 COM-segment [9].
As recommended by the JPEG2000 standard, the COM segment should only be used to embed informative information, such as software version or any copyright information. Furthermore, they explicitly recommend not to use the COM-segment for embedding any data necessary to decode or properly interpret a JPEG2000 encoded image. Hence, neither any meta data required to properly interpret and exploit the compressed imagery nor any information that might be used to improve JPEG2000 decoder performance shall be placed within a COM-segment [9].
Although embedding decoding information into the JPEG2000 COM-segment is not recommended by the JPEG2000 standard, this work will employ it as an additional method to embed the encryption specification into the JPEG2000 codestream. However, the risk of losing the embedded data is high, as removing the meta data or converting the image to another file-format will delete the encryption specification completely (if no intermediate processing step to extract and to embed the data into the other file are conducted).

4.3.2 Non-Format-Compliant Data Embedding

A non-format-compliant way to losslessly embed the encryption specification into the JPEG2000 codestream is to either insert the data at the very beginning of the file or at its end. Therefore, embedding the encryption specification prior the first marker is achieved by embedding the data right before the *Start of Codestream* (SOC) marker, which is indicated by the value 0xFF4F in the JPEG2000 codestream. The encrpytion specification is embedded after the *End of Codestream* (EOC) marker, which is indicated by the value 0xFFD9. The SOC- and the EOC-marker are both required and should only occur once in the JPEG2000 codestream [9].
However, adding data in this way is not JPEG2000 format-compliant, as the standard specifies that the JPEG2000 codestream must start with the SOC-marker and end with the EOC-marker. Furthermore, as pointed out by Engel et al. [58], special care must be taken in order to not insert additional marker segments. Hence, payload bytes exceeding 0xFF8F must be escaped. Otherwise, they would be interpreted as additional markers.

Depending on how escaping is done, this could lead to additional overhead. Due to the fact that embedding the encryption specification in this way is not JPEG2000 format compliant, most image viewers and editors will not be able to open files edited in this way [58]. This non-format-compliant binary data embedding method has the advantage of being computational less demanding and it poses no risk for degrading the perceptual image quality, as no JPEG2000 image coefficients are modified.

4.3.3 Length-Preserving Data Embedding

Finally, the third embedding approach evaluated by this work embeds the encryption specification into the JPEG2000 bitstream by substituting JPEG2000 image coefficients with the encryption specification. Hence, by using this method, a JPEG20000 standard compliant decoder is still able to decode the image without any problems.
As mentioned above, this embedding approach embeds the encryption specification into the JPEG2000 codestream by overwriting certain parts of the bitstream. Therefore, for not overwriting the image coefficient belonging to the RoI, we have decided to embed the encryption specification at the very end of the JPEG2000 codestream, just before the EOC-marker concludes the codestream. This bitstream location has been selected due to the fact, *quality progression bitstream ordering* (as mentioned in previous chapters), which implies that all data packets belonging to an RoI are aligned at the beginning of the codestream.
Embedding the encrypting-specification into JPEG2000 codestream by replacing image coefficients leads to numerous benefits, such as length-preservation, format-compliance and efficient data embedding. However, this simple and fast data embedding approach has one drawback, one of altering the perceptual image quality. The degree to which the embedded data influences the perceptual image quality is evaluated in Chapter 7.

Approach	Format-compliance	Losslessness	No Overhead
COM-Segment	✓	✓	×
Before-SOC-Marker	×	✓	×
After-EOC-Marker	×	✓	×
Data Embedding	✓	×	✓

Table 4.1: This table lists the proposed embedding approaches and their characteristics concerning format-compliance, image-quality (losslessness) and overhead (length-preservation), which indicates whether the length of the JPEG2000 codestream has been changed while embedding the encryption specification.

Chapter 5

Automated RoI Detection

This chapter provides an overview of the automated RoI detection methods we have proposed. We have identified five different RoI detection approaches, with which it may be possible to detect the encrypted image region accurately.
The first three methods proposed by this work are concerned with detecting the RoI by its entropy, variance or by a defined threshold. For all three scenarios, the image containing the encrypted image region needs to be spitted into smaller non-overlapping rectangular image blocks. These image blocks are further used to calculate the variance/entropy or whether the image block contains any image value exceeding the threshold.The final two methods proposed by this work rely on detecting the encrypted image region by using edge detectors. Therefore, the Sobel- and the Canny-edge-detector have been chosen to detect the edges within an image.

5.1 Detect RoI by Entropy

In information theory, the concept of Shannon's entropy is used as a measure of uncertainty in a random variable [59]. The entropy of a random variable is defined in terms of its probability distribution and can be shown to be a good measure of randomness or uncertainty [60]. Therefore, this section describes the fundamentals of entropy and the reasoning, why we have chosen the entropy to detect the encrypted image region.

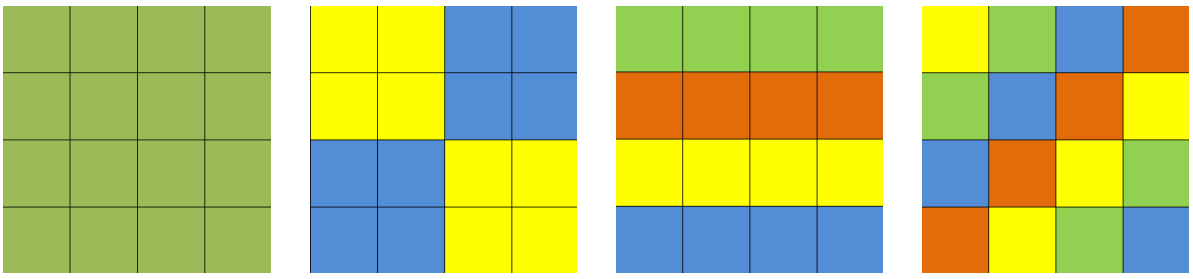
5.1.1 Definition

Shannon denoted the entropy H of a random variable X with a finite number of possible values $x_1, x_2, \ldots, x_n$ and with probabilities $p(x_i)$ respectively such that $p(x_i) \geq 0, i = 1, 2, \ldots, n \sum_{i=1}^{n} = 1$ [61].

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{5.1}$$

5.1.2 Reasoning for Choosing the Entropy

As pointed out by Wu et al.[62], an encrypted image has a random-like image characteristic. Because of this, the entropy of an encrypted image region needs to be higher than in the unencrypted image regions. Figure 5.1 shows the effect of encrypting a source image on the entropy. Therefore, as evident from the figures below, the entropy is low in unencrypted images (see Figure 5.1a) and increases with random-like images (see Figure 5.1d).



(a) (b) (c) (d)

Figure 5.1: The figures above show some Entropy examples: Figure 5.1a - $H = 0{,}0\,bits$, Figure 5.1b - $H = 1{,}0\,bits$, Figure 5.1c $H = 2{,}0\,bits$, Figure 5.1d - $H = 2{,}0\,bits$,

5.2 Detect RoI by Variance

In probability theory and statistics, the variance is a measure of how far a set of numbers is spread out. In other words, the variance describes how far the numbers (e.g., luminance values of a pixel) lie from the mean value μ (i.e. average of all pixel values) of the data set. Therefore, the variance is a mathematical expectation of the average squared deviations from the mean. Hence, the variance has the advantages of mathematical and computational simplicity compared to other probability distribution descriptors [63].

5.2.1 Definition

The variance of X with a finite number of possible values $x_1, x_2, \ldots, x_n$ and with probabilities $p(x_i)$ respectively such that $p(x_i) \geq 0, i = 1, 2, \ldots, n \sum_{i=1}^{n} = 1$ is given by:

$$Var(X) = \sigma^2 = \sum_{i=1}^{n} p(x_i) \cdot (x_i - \mu)^2 \tag{5.2}$$

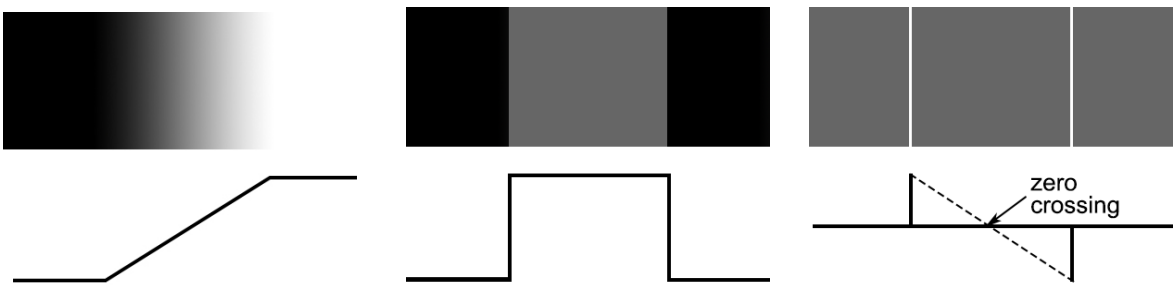
5.2.2 Reasoning for Choosing the Variance

Based on the assumption that an encrypted image region should have an increased variance, we have chosen the variance to detect the encrypted image region automatically. This assumption is based on the fact that the encryption algorithm, used to encrypt the

image region, spreads the pixel values over the whole luminance range (e.g., in an 8 bit image, the luminance value ranges from 0 to 255). Therefore, the probability that two pixel located next to each other differ in their luminance value by several orders of magnitude is more likely than in an unencrypted image. Hence, distinguish between unencrypted- and encrypted-image regions should be possible.

5.3 Use Edge Detector to Detect RoI

Edge detection is an image processing technique used to mathematically detect the pixels in digital images at which the image brightness changes sharply. In other words, the objective of these methods is to detect edges by a strong intensity of contrast - a jump in intensity from one pixel to the next. Therefore, edge detectors significantly reduce the amount of image data and filters out useless information (data not belonging to an edge), while preserving the important structural properties in a digital image [15, 64].
As stated by Green [15], there are two main edge detection categories used to categorise the majority of different edge detection methods. These categories are gradient and Laplacian. The gradient methods determine the location of edges by looking for maximum and minimum in the first derivative of the image, whereas the Laplacian methods use the zero crossings in the second derivative of the image to detect edges.
As depicted in Figure 5.2a, the edge has the one-dimensional shape of a ramp and, by calculating the derivative of the source image, can highlight its location. Therefore, Figure 5.2b represents the first derivative (used by the gradient methods) and Figure 5.2c shows the second derivate (used by the Laplacian methods) of the source signal.



(a) Edge Detector – Input Signa [65]l

(b) Edge Detector – First Derivative [65]

(c) Edge Detector – Second Derivative [65]

Figure 5.2: The figures above show the effect of applying the first- or the second-derivative to a source signal. Figure 5.2a shows a ramp edge, which is used as source signal, to calculate the 1st- and 2nd derivative. Figure 5.2b shows the first derivative. This approach is used by the gradient methods (e.g., Canny, Sobel). Figure 5.2c depicts the second derivative, which is used by the Laplacian edge detection methods to detect edges by looking for zero crossings [65].

5.3.1 Sobel Edge Detection

Similar to the Canny algorithm, the Sobel Edge Detection operator is used to detect image-edges. In order to fulfill this objective, the Sobel operator measures the 2-D spatial image gradient. This is done by applying a pair of 3×3 convolution masks, one estimating the gradient in the x-direction (used for the columns) and the other estimating the gradient in the y-direction (used for the rows). The convolution masks are depicted in Figure 5.3, whereby the two masks differ from each other by simply rotating one by 90° .

-1	0	1
-2	0	2
-1	0	1

(a)

1	2	1
0	0	0
-1	-2	-1

(b)

Figure 5.3: Masks used by Sobel Edge Detector: G_x (a) and G_y (b) [15]

As the convolution masks are usually much smaller than the actual image, the masks are slid over the image, manipulating only one pixel at a time. These masks are designed to respond best to edges running vertically and horizontally relative to the pixel grid. The property of detecting vertical and horizontal edges best represents in conjunction to this work an advantage, as all the encrypted RoIs are limited to vertical and horizontal edges. As pointed out by Raman and Himanshu [64], the masks (G_x and G_y, as depicted in Figure 5.3) can be applied separately to the source image. Thereby they produce separate measurements of the gradient component in each orientation. Hence, the measurements can be combined in order to determine the absolute magnitude of the gradient at each point. The gradient magnitude is given by:

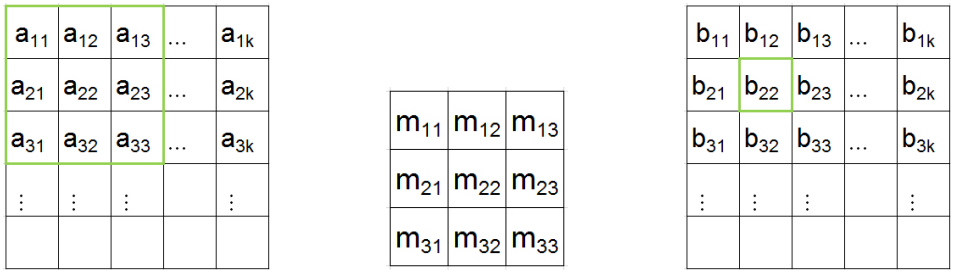
$$|G| = \sqrt{G_x^2 + G_y^2} \tag{5.3}$$

However, due to performance reasons, an approximation of the gradient magnitude is typically computed.

$$|G| = |G_x| + |G_y| \tag{5.4}$$

As mentioned above, the mask is slid over an area of the source image and manipulates only one pixel at a time. After changing that pixel's value, the mask shifts one pixel to the right and continues to the right until it reaches the end of the row. When the end of the first row is reached, the mask continues at the second row to manipulate the pixel's value and shifts one pixel to the right and so on, until the end of the image is reached. Figure 5.4 depicts how the mask is slid over the source image and how the output pixel's values are computed. Therefore, the image region, which is processed by the Sobel mask,

and its corresponding output pixel are highlighted by a green border. Formula 5.5 is used to calculate the output pixel.



(a) Source Image **(b)** Mask **(c)** Output Image

Figure 5.4: This figure demonstrates how the Sobel mask is slid over an input image and how the pixel values are changed (see Formula 5.5)

Formula 5.5 shows how a particular pixel in the output image would be calculated. Therefore, the center of the mask is placed over the pixel, which will be manipulated. However, it should be noted that all the pixels located at the image borders (first-, last-column and first-, last-row) cannot be used by the Sobel edge detector, due to the fact that the 3×3 mask cannot place its center over these pixels.

$$\begin{aligned} b_{22} = (a_{11} \cdot m_{11}) &+ (a_{12} \cdot m_{12}) + (a_{13} \cdot m_{13}) + \\ (a_{21} \cdot m_{21}) &+ (a_{22} \cdot m_{22}) + (a_{23} \cdot m_{23}) + \\ (a_{31} \cdot m_{31}) &+ (a_{32} \cdot m_{32}) + (a_{33} \cdot m_{33}) \end{aligned} \tag{5.5}$$

5.3.2 Canny Edge Detection

Although the Canny edge detector is more complex than the Sobel edge detector, its superior performance might result in a superior encryption detection. The superior performance results are based on the fact that the Canny edge detector achieves a very low error rate and the localization of the edge points is done very accurately. In other words, the Canny Edge Detection algorithm is able to differentiate between edges and non-edges with a low error rate. Furthermore, the distance between the edge pixels determined by the detector and the actual edge is at a minimum. In order to achieve this, the Canny edge detector combines a number of techniques to detect and refine edge decisions. The main processing steps are highlighted in the following [15, 64]:

1. The first step handles the noise reduction. Therefore, the source image is filtered by a Gaussian filter. Due to the fact that the Gaussian smoothing mask (similar to the Sobel mask) is typically much smaller than the actual image, the mask is slid over the source image, manipulating a square of pixel at a time. Hence, the larger the

Gaussian mask is, the lower the detector's sensitivity is to noise. The filtering step results in a smoothed image, which contains less noise.

2. After smoothing the image and reducing the noise, the gradient magnitude of the edges is calculated. Therefore, the Sobel masks G_x and G_y, as depicted in Figure 5.3, can be used to calculate the gradient magnitude (edge strength) at each point.

3. Next, the edge directions are calculated. Therefore, as outlined by Green [15], the edge direction is calculated as follows: $\theta = \arctan(\frac{G_x}{G_y})$, where G_x and G_y are the gradient scores respectively, calculated by the two Sobel masks.

4. The next step in detecting the image edges accurately, is to relate the edge direction to a direction that can be traced in an image. Therefore, the computed direction θ needs to be rounded to one of four valid image directions 0°, 45°, 90° or 135°. Obviously, edges resulting in 180° are mapped to 0°, 225° = 45°, etc. This means if θ ranges between $[-22{,}5° \cdots 22{,}5°]$ or $[157{,}5° \cdots 202{,}5°]$, the edge direction would be rounded to 0°. Figure 5.5 depicts each edge direction in a different color. The colors would repeat on the lower half of the circle.

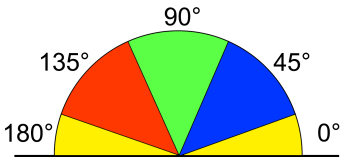


Figure 5.5: Canny Edge Detector – Possible Edge Directions [15]

5. Then, the non-maximum suppression is applied to the detected edges. This step is performed due to the fact that the edges detected by the Sobel masks can either be very thick or very narrow, depending on the intensity across the edge and how much the image was blurred at first. Therefore, the non-maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value considered not to be an edge. Hence, only the edges with the highest gradient magnitude are kept, which results in a thin edge line.

6. Finally, the hysteresis is used to eliminate the effect of streaking edges. Therefore, two thresholds are used, a high T1 and a low T2. Hence, if any pixel in the image exceeds T1, it is presumed that this pixel belongs to an edge. Then, any pixels that are connected to this edge pixel and that have a value greater than T2 are also selected as edge pixels.

A more thorough explanation of the Canny algorithm can be found in the paper, "A Computational Approach to Edge Detection" [66].

Chapter 6

Implementation of JPEG2000 RoI-Detection-Methods

This chapter gives detailed information about the implementation of the embedding- and automated-RoI detection methods proposed by this work. All investigated approaches are based on Stubhann's implementation of a JPEG2000 RoI bitstream encryption [5], which follows an approach proposed by Hämmerle-Uhl et al. [19]. To be precise, the "Max-Shift RoI encryption" method (see Section 3.3) has been used to evaluate the results regarding computational demands, impact on image quality, JPEG2000 format-compliance and real-world feasibility. Due to the fact that Stubhann's implementation is based on the open-source JPEG2000 Part 1 Java implementation JJ2000[1], which is available under GNU Lesser GPL type licensing, all functions we have implemented are also written in Java. Furthermore, we need to highlight that this chapter focuses exclusively on the implementation details and design decisions. Hence, no in-detail explanation of any previous implementations (e.g., JJ2000 or Stubhann's bitstream encryption) is given.

6.1 Development Environment

This implementation is realized using the Eclipse Indigo Enterprise Edition Software Development Kit Version 1.4.2 in connection with the Java Standard Edition Development Kit (JDK) version 1.7.0_03. All experiments carried out to evaluate the embedding- or the automated RoI detection methods are based on the image database SCFace [46]. This is due to the fact that we wanted to compare the embedding overhead caused by the proposed embedding methods with Stubhann's experimental results [5]. Furthermore, this image database offers a wide range of different images. To be more precise, the SCFace image database provided us with video-surveillance camera images of 130 persons (stored in JPEG file-format). Furthermore, the SCFace image database contains, in addition to the image files, a textile incorporating the x- and y-coordinates of the eyes, tip of the nose

[1] http://jj2000.googlecode.com/svn/trunk/

and the mouth. This text-file was used by Stubhann to determine the Region of Interest (the facial area) of the people being recorded by the video-surveillance systems. Consequently, he was able to determine the image region covering the facial area of the recorded people and to successfully encrypt certain bitstream parts belonging to the determined RoI. For further details on how Stubhann handles the RoI detection and its encryption, reference is made to Stubhann's Master Thesis [5].

6.2 Data Embedding Techniques

As mentioned in Section 2.2, there are five ways of ordering the JPEG2000 codestream. However, in connection with embedding the encryption specification into the JPEG2000 codestream, this work focuses exclusively on the *quality progression bitstream ordering*. Hence, all data packets belonging to an RoI or multiple RoIs are aligned at the beginning of the JPEG2000 codestream. Therefore, this type of ordering the JPEG2000 bitstream offers the advantage of reducing the information required to specify the encrypted codestream parts. Hence, it is sufficient to store the length, in bytes, of the encrypted RoIs within the codestream and their encryption counters, in order to successfully decrypt the encrypted bitstream data packets. This is based on the fact that the encryption length reveals the end of the encrypted bitstream, which is used to extract solely the encrypted packets. Furthermore the encryption counters are used, to signal how often packets are encrypted. This encryption-counter is required due to the multiple packet encryption, which is used due to the fact that applying multiple encryption avoids the creation of additional marker segments (see Subsection 3.3.4). However, the encryption counter is only embedded into the codestream if a packet is encrypted more than once. Hence, JPEG2000 packets which are encrypted only once are not embedded into the JPEG2000 codestream. Because of this, the embedding overhead can be reduced to a minimum, as not all the encrypted packets need to be embedded into the JPEG2000 codestream (see Chapter 7). Therefore, this section describes the implementation details and the proposed packet structure used to embed the encryption specification into the JPEG2000 codestream.

6.2.1 Embed Data into the JPEG2000 COM-Segment

The JPEG2000 COM segment, as described in Section 4.3.1, can be used to store up to 65530 bytes of additional payload into the JPEG2000 codestream. Therefore, the COM segment offers the possibility to store the specification of the encrypted RoI bitstream parts into the JPEG2000 codestream (length of encrypted bitstream and encryption counter per packet). Figure 6.1 depicts the proposed packet structure for embedding the

encryption specification into the JPEG2000 COM segment. It starts with the COM-marker (0xFF64), which indicates the start of the COM segment. The COM-marker is then followed by the Lcom, which gives information about the length of the COM segment (the length is at least 8 bytes, when data embedding has been applied). The Lcom is calculated as follows: 2 bytes for the Lcom itself, 2 bytes for the Rcom, 4 bytes for the length of the encrypted codestream and if a packet is encrypted multiple times, 3 bytes must be added for each multiple encrypted packet. The Rcom (registration values) succeeds the Lcom, which indicates the type of data used in the COM segment. The Rcom can assume the value 0 or 1, whereby 0 indicates that any binary data is allowed within the COM segment and 1 indicates that only text encoded bytes are allowed. Finally, the payload field of the COM segment is used to embed the encryption specification. As depicted in the following figure, the packet-structure starts by the length of the encrypted bitstream and continues with the packet-IDs and their corresponding encryption counters (how often the packet has been encrypted). However, the packet-ID and its corresponding encryption counter is is only added if the packet is encrypted more than once. Adding only packets which have been encrypted multiple times offers the advantage, of superior embedding overhead (less overhead), as experimental results have shown.

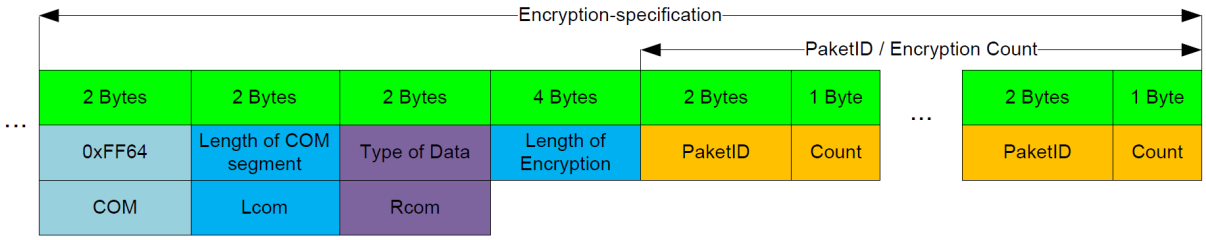


Figure 6.1: Packet structure used to embed the encryption specification into the JPEG2000 COM-segment

The basic embedding procedure, used to embed the encryption specification into the JPEG2000 COM-segment, looks as follows:

1. Load encrypted bitstream from hard drive
2. Detect position of COM marker (0xFF64) in encoded and partly encrypted JPEG2000 bitstream (see Source-code A.2)
3. Create payload, containing the length of the encrypted bitstream and if a packet is encrypted multiple times, the packet-ID + its encryption counter
4. Replace the Lcom field, with the new length of COM segment (2 bytes for Lcom + 2 bytes for Rcom + size of payload); See Source-code A.3
5. Finally, embed the encryption specification into bitstream (see source-code A.1)

The extraction of the embedded RoI-information is performed quite similarly to the procedure outlined above. At first, the position of the COM-marker within the encrypted bitstream must be detected. Afterwards, the Lcom (length of COM segment) and the length of the encrypted bitstream are read. The Lcom is used to determine the number of packets, which have been encrypted multiple times, as it gives information about the length of the COM segment. Finally, the packet-IDs and their corresponding encryption counters can be read (see source-code A.4). After extracting the encryption specification completely, the encrypted JPEG2000 data packets can be decrypted.

6.2.2 Embed Data prior to the JPEG2000 SOC-Marker

As outlined in Subsection 4.3.2, embedding the encryption specification prior to the encoded JPEG2000 codestream is a non-format-compliant way of embedding data into the JPEG2000 codestream. Hence, a standard compliant JPEG2000 decoder might not be able to decode the JPEG2000 codestream properly.

Figure 6.2 depicts the proposed packet structure for embedding the encryption specification prior to the JPEG2000 SOC-marker. The designed packet structure starts with the so called *Active Encryption Marker*, which has been introduced by us to signal that data has been embedded into the JPEG2000 codestream. Therefore, the byte value 0xFF8E has been chosen to represent the *Active Encryption Marker*. This is based on the fact that this byte value represents the lowest value not in use by the standard JPEG2000 coding system to signal a marker. The *Active Encryption Marker* is followed by the length of the encrypted JPEG2000 codestream (length of all the encrypted RoI image coefficients, stated in bytes). Finally, the packet-IDs and their corresponding encryption counters are added for all packets which are encrypted more than once. Adding only packets which have been encrypted multiple times offers the advantage of superior embedding overhead (see Chapter 7), as experimental results have shown. Although embedding the encryption specification prior to the JPEG2000 codestream is straight forward, the impact of non-format compliance on standard compliant JPEG2000 decoders needs to be investigated.

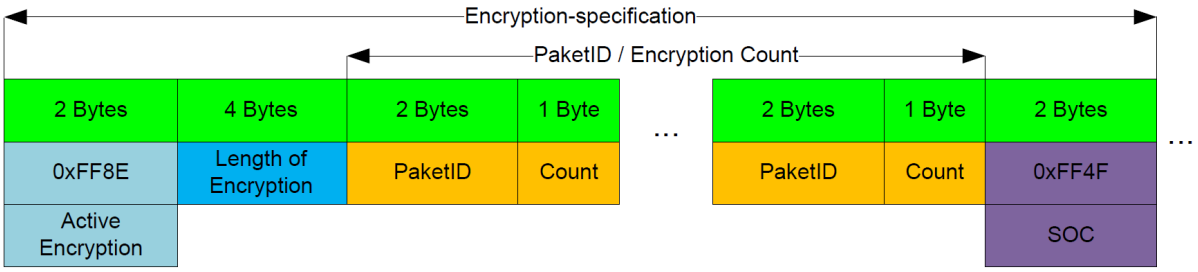


Figure 6.2: Packet structure used to embed the encryption specification prior to the JPEG2000 SOC-marker

The basic embedding procedure used to embed the encryption specification prior to the SOC-marker looks as follows:

1. Load partly encrypted bitstream from hard drive
2. Copy bitstream into byte array (see Source-code A.5, line 4-7)
3. Create encryption specification, containing the length of the encrypted bitstream and if a packet is encrypted multiple times, the packet-ID + its encryption counter
4. Write encryption specification to output file (see Source-code A.5, line 9-14)
5. Write temporary byte array to output file (see Source-code A.5, line 15-16)

The extractionprocedure that we have implemented starts by checking if the JPEG2000 codestream starts with the *Active Encryption Marker* (0xFF8E), which has been introduced to signal active data embedding (see source-code A.6). Subsequently, the length of the encrypted bitstream is extracted from the codestream. Then the position of the SOC-marker in the JPEG2000 bitstream is identified, which is required to determine how many packets are encrypted more than once. After extracting the length of the encrypted bitstream and the position of the SOC-marker within the JPEG2000 codestream, the remaining encryption specification can be read from the codestream (information about packets encrypted more than once). Thereafter, the embedded data should be removed from the JPEG2000 codestream, as otherwise a standard compliant JPEG2000 decoder might not be able to decode the image properly. This is due to the fact that the codestream does not start with the expected SOC-marker. Finally, using the encryption specification, the encrypted JPEG2000 packets can be decrypted.

6.2.3 Embed Data after the JPEG2000 EOC-Marker

As outlined in Subsection 4.3.2, embedding the encryption specification after the encoded JPEG2000 codestream is a non-format-compliant way of embedding data into the JPEG2000 codestream. Hence, a standard compliant JPEG2000 decoder might not be able to decode the JPEG2000 codestream properly.
Figure 6.3 depicts the packet structure proposed to embed the encryption specification after the EOC-marker. The designed packet structure differs from the packet structure designed for embedding the data prior to the SOC-marker solely in the position of the *Active Encryption Marker*, which is aligned after the embedded encryption specification. This design decision has been made due to the fact that adding a delimiter to the embedded data structure enables the detection of whether the codestream has been fully loaded or if the codestream has been damaged while transferring/storing it.

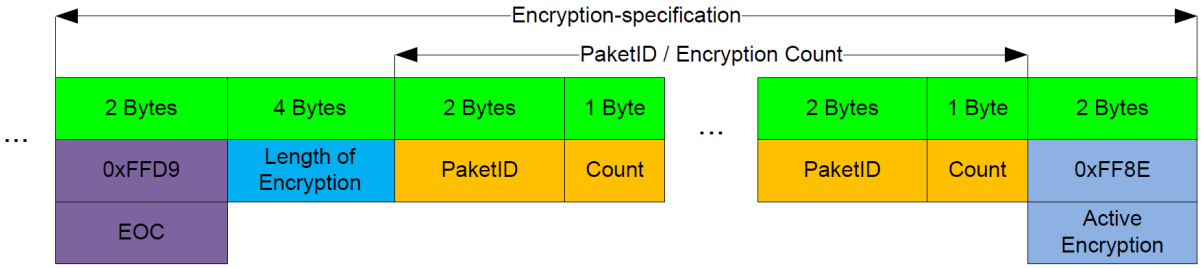


Figure 6.3: Packet structure used to embed the encryption specification after the JPEG2000 EOC-marker

Embedding and extracting the encryption specification after the EOC-marker are both performed as outlined in Subsection 6.2.3. However, embedding the encryption specification after the EOC-marker slightly differs from embedding the encryption specification prior to the SOC-marker. The first difference is obviously the bitstream position used to embed the encryption specification, which has been changed from the beginning of the JPEG2000 codestream to its end. The second difference is the position of the *Active Encryption Marker*, which is located at the end of the JPEG2000 bitstream. Finally, after extracting the encryption specification and removing it from the codestream, the partly encrypted JPEG2000 bitstream can be decrypted by using the gathered encryption specification and decoded by a JPEG2000 compliant decoder.

6.2.4 Length-Preserving Data Embedding

As outlined in Subsection 4.3.3, embedding encryption specification by replacing JPEG2000 image coefficients has the advantage of being JPEG2000 format-compliant and resistant against removing the meta data, as no JPEG2000 file-format specific container has been used to embed the encryption specific information. Furthermore, the length of the codestream is not changed, which might be required by some applications.

Figure 6.4 depicts the packet structure used to embed the encryption specification into the JPEG2000 codestream. The designed packet structure starts with the so called *Active Encryption Marker* (0xFF8E), which we have introduced to signal that data has been embedded into the JPEG2000 codestream. This marker is followed by the length of the encrypted JPEG2000 codestream (length of all the encrypted RoI image coefficients, stated in bytes). Finally, the packet-ID and its corresponding encryption counter for all packets encrypted more than once are embedded into the JPEG2000 codestream. The EOC-marker (0xFFD9) concludes the encoded encryption specification and the JPEG2000 codestream. However, it should be noted that, while embedding the encryption specification, no JPEG2000 markers should be modified or added, as this would violate JPEG2000 format-compliance. The decision for embedding the encryption specification prior to the

EOC-marker by replacing the JPEG2000 image coefficient is based on the fact that the *Image quality Progression Bitstream ordering* has been used while encoding the image, as mentioned in previous sections. Hence, all RoI relevant image coefficients (encrypted coefficients) are aligned at the beginning of the codestream, and the image coefficients at the end of the codestream contain only background information.

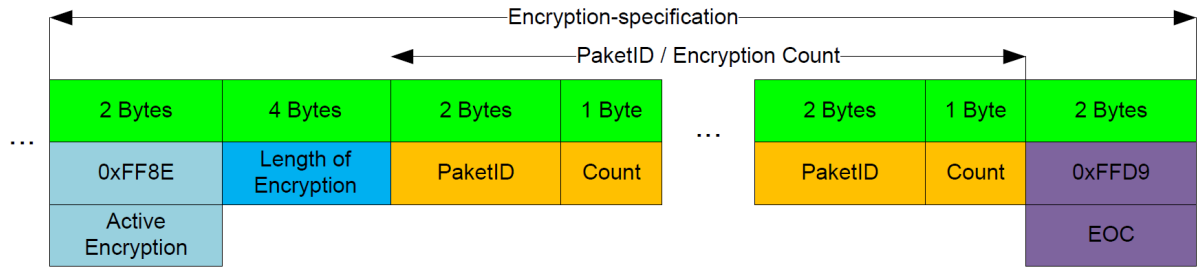


Figure 6.4: Packet structure used to embed the encryption specification into the JPEG2000 codestream by replacing JPEG2000 image coefficients

The basic embedding procedure, used to embed the encryption specification into the JPEG2000 codestream by replacing image coefficients, looks as follows:

1. Load partly encrypted bitstream from hard drive
2. Create encryption specification, containing the length of the encrypted bitstream and if a packet is encrypted multiple times, the packet-ID + its encryption counter
3. Identify position for inserting encryption specification (Length of encoded and encrypted JPEG2000 codestream - 2 bytes (EOC-marker) - n-bytes for length of multiple encrypted packets - 2 bytes for length of encrypted codestream, used to encrypt the RoI coefficients - 2 bytes for *Active Encryption Marker*). During this step, one must be careful of the overlapping with any existing JPEG2000 markers.
4. Write *Active Encryption Marker* to output file (see Source-code A.7)
5. Write encryption specification to output file

The extraction of the embedded encryption specification starts by determining the position of the *Active Encryption Marker* (0xFF8E) within the JPEG2000 encoded bitstream. If the *Active Encryption Marker* has been found, the length of the encrypted bitstream is extracted, which is located next to the *Active Encryption Marker*, as described above. Thereafter, the length of the embedded encryption specification is calculated (for more details see Source-code A.8), which is used to extract the embedded encryption specification. Finally, after extracting the embedded data, the JPEG2000 bitstream can be decrypted by using the gathered encryption specification.

6.3 Automated RoI Detection

This section describes the implementation details of the methods proposed in Chapter 5. Basically, the idea behind these methods is to automatically detect the encrypted RoI (e.g., facial area of a person who is recorded by a video-surveillance system) without any additional information. Therefore, the methods in this section differ from the methods proposed in the previous section only by not storing the encryption information into the JPEG2000 codestream (see Section 6.2 for further implementation details). Therefore, this section is divided into two main parts. The first part presents the three ways of acquiring the image data (e.g., from: JPEG2000 packet data, PGM image data, and inverse Wavelet coefficients) and the second part describes the implementation details when it comes to automatically detecting the encrypted image parts.

6.3.1 Acquiring the Image-Data

In order to apply the automated RoI detection methods, the image data must be acquired. This work has proposed the following ways of acquiring the data:

From PGM-File:

In order to acquire image data from a PGM-file (portable gray map format), the JPEG2000 image must be decoded and stored as PGM-file. Afterwards, the PGM file is loaded and partitioned into smaller non-overlapping rectangular blocks of equal size (except those located at the image boarders). Partitioning the image into smaller blocks enables the automated RoI detection methods to detect differences in variance, entropy, etc., for example, hence detecting the encrypted image region. Figure 6.5 shows the partitioning of the PGM file in smaller non-overlapping rectangular blocks, which are used to automatically detect the encrypted image region.

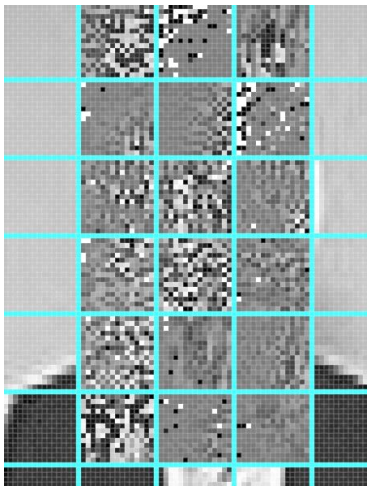


Figure 6.5: Partitioned PGM Input Image

From JPEG2000-Codestream:

Acquiring image data from the JPEG2000 codestream is performed as follows: First, the JP2 image-file is loaded (this file contains the encrypted JPEG2000 codestream). Next, the codestream is parsed accordingly, which implicates the detection of the JPEG2000

packet bodies (encapsulated by the SOP- and the EPH-marker, as depicted in Figure 2.8). Finally, the automated RoI detection methods are applied to the packets.

From JPEG2000-Decoder:

The last image data acquisition method is concerned with extracting the image-data while decoding the JPEG2000 image. Therefore, all the JPEG2000 decoding steps (outlined in Chapter 2) are executed until the inverse Wavelet transformation has been applied. Extracting the image data from the inverse Wavelet transformation is based on the fact that the Wavelet image coefficients provide the best encryption detection performance, as experimental results have shown. After extracting the inverse Wavelet coefficients from the JPEG2000 decoder, the data is partitioned into smaller code-blocks of equal size (comparable to Figure 6.5). However, selecting the code-block-size used to partition the extracted image data should not be done randomly, as the best results can be achieved when the same code-block-size is selected as the size used to encode the JPEG2000 image. Hence, we extract the code-block-size used to encode the JPEG2000 image from the JPEG2000 decoder. Finally, the partitioned Wavelet data is forwarded to the automated RoI detection methods, which are explained in more detail in the following subsections.

6.3.2 Detect RoI by Entropy

The first automated RoI detection method relies on the entropy to detect the encrypted image region. Therefore, as outlined by the data acquisition methods, the source data is further partitioned into smaller parts (image block, or JPEG2000 packet body data), which are used to calculate the entropy. These smaller image-parts, be it from a PGM-file, the JPEG2000 packet data or from the inverse JPEG2000 Wavelet transformation, are henceforth called blocks. After extracting the data from any of the three input sources and partitioning it into smaller blocks, the data are forwarded to the entropy calculator, which calculates the entropy for each block (Source-code B.6 depicts a code snippet to load and parse the JPEG2000 codestream).
Subsequently, as pointed out by Wu et al.[62], it should be possible to detect the encrypted image region. This assumption is based on the fact that an encrypted image has a random-like image characteristic, hence the entropy of these image parts needs to be higher than in the other unencrypted parts of the image [62]. Source-code B.1 depicts the implementation of the entropy-calculation, which features a histogram function B.3 to calculate the entropy value. The histogram function is used to calculate the distribution of the input-values. For further details about how the entropy is calculated or its formula, reference is made to Section 5.1.

6.3.3 Detect RoI by Variance

The second automated RoI detection method relies on the variance to detect the encrypted image region accurately. Similar to the entropy image encryption detection method, this method calculates the variance on parts of the source image at a time (partitioned image, or JPEG2000 packet body data). This blocks represent a smaller part of the source image, as calculating the variance for the whole image would be useless. This is due to the fact that no distinction between an unencrypted image region and an encrypted image region could be made, as only one variance calculation has been performed.
After calculating the variance values (of each image block) and defining a proper threshold, the encrypted image regions should be detectable. The threshold is used to distinguish between an encrypted and an unencrypted image block, as the encrypted image regions should have an increased variance value.
Source-code B.4 depicts the implementation of the variance-calculation, which features the histogram function B.3 to calculate the variance value. The histogram function is used to calculate the distribution of the input-values. For further details about how the variance is calculated or its formula, reference is made to Section 5.2.

6.3.4 Detect RoI by Thresholding

Thresholding is the simplest approach to determine the encrypted image region. This is due to the fact that it is sufficient to simply check whether the values extracted from the JPEG2000 inverse Wavelet-transformation are valid image values. The Thresholding decision boundary, as implemented by this work, looks as follows:

$$\begin{array}{llll} \text{if} & T^U < f(x,y) > T^L & \text{then} & f(x,y) = \textbf{Encrypted content} \\ \text{else} & & & f(x,y) = \textbf{Unencrypted content} \end{array}$$

Where x and y define the pixel at a specific image location, T^U and T^L represent the upper-, respectively the lower-threshold. In our case, the upper-threshold is 255 and the lower-threshold is 0, which is based on the fact that we used an 8 bit source image. Furthermore, the signed source image values (ranging from -128 to 127) converted to an unsigned version (ranging from 0 to 255), as depicted in Source-code B.6 at line 19. Hence, all valid image values should range from between 0 and 255.
However, detecting the encrypted image region by thresholding is limited to the values extracted from the JPEG2000 inverse Wavelet-transformation. As the other data acquiring methods described above provide either values which are already in a valid range (load from PGM-file) or are due to the JPEG2000 entropy encoder, random like distributed.

Hence, no proper threshold could be defined. The results of checking whether the image block contains any invalid values are stored in a simple Boolean array (see Thresholding method Source-code B.2), which is forwarded to the bounding-box calculation method (see Source-Code B.5), which calculates the bounding box of all the blocks containing an invalid number. Figure 6.6 depicts a sample bounding-box of blocks containing invalid numbers, whereby the grid visualizes the partitioning of the Wavelet-coefficients and the black filled rectangles represent the blocks containing an invalid number. The dotted border visualizes the calculated bounding box, which represents the border of the detected encrypted image region. As the figure illustrates, not all blocks contain invalid numbers, even if they are encrypted. However, due to the fact that the bounding-box is calculated in order to surround all invalid image blocks, accurate encryption detection can be achieved. However, this approach, as well as all the other automated encryption detection methods we have proposed, are limited to detecting only one encrypted RoI per image, which might not be accepted by all applications.

Figure 6.6: Bounding-Box – Thresholding

6.3.5 Detect RoI by Canny- or Sobel-Edge-Detector

Detecting the RoI by either applying the Canny- or the Sobel-Edge-Detector is performed on the whole image, which differs from the other automated RoI detection proposed methods implemented by this work (which were performed on block basis). This is due to the fact that detecting the edges of one code-block at a time would be useless, as the edges of the whole image are needed to detect the encrypted RoI. Furthermore, it is important to notice that all pixels at the borders (first-, last-column and first-, last-row) cannot be used by the Sobel- or Canny-edge-detector, due to the fact that the 3×3 mask, used to calculate the edge strength (gradient magnitude), cannot place its center over these pixels. Therefore, images containing encrypted RoIs bordering the image borders might not be detectable by the proposed edge detectors.

The edge-detectors proposed by this work cannot use all three input resources implemented by this work, as the data acquired by extracting the JPEG2000 packet data provides no information which could be used to detect any borders. This is due to the fact that the encoded image coefficients stored in JPEG2000 packets feature no image edge characteristics which could be used by the edge detectors.

Chapter 7

Performance Evaluation

This chapter presents and discusses the results of the proposed embedding- and automated RoI detection approaches outlined in previous chapters. Therefore, this chapter is divided into three main sections. The first section describes the experimental setup, which has been used to evaluate the proposed methods. The second section presents and discusses the results of the proposed embedding methods. Finally, the third section presents and discusses the experimental results obtained by the proposed automated RoI detection methods.

7.1 Experimental Setup

This section gives an overview of the equipment and the development environment used to evaluate the proposed embedding- and automated RoI detection methods.

The hardware and software used for the experiments is listed below:

- **Laptop:** Dell XPS L502X
- **CPU:** Intel Core i7-2670QM at 2,2 GHz
- **RAM:** 2×4GB DDR3-1333
- **Display:** NVIDIA GeForce GT 540M
- **Hard drive:** SSDSA2CW160G310 320 Series SSD 160GB, SATA 300
- **Operating System:** Windows 7 Professional 64-bit, Service Pack 1
- **Development environment:** Eclipse Indigio Enterprise Edition Version 1.4.2
- **Standard Development Kit:** Java JDK Version 1.7.0_03
- **JPEG2000 Coder:** JJ2000[1] Version 5.1

[1] http://jj2000.googlecode.com/svn/trunk/

7.2 Data Embedding Techniques

This section presents and discusses the results of the proposed data embedding techniques. Therefore, the experimental measurements are evaluated for their suitability based on the criteria outlined in Section 3.2. The criteria used in this section are Format-Compliance, Overhead, Computational-Demand and Image-Quality. The other two criteria mentioned in Section 3.2 are not covered in this section, due to the fact that, on the one hand, the Security criterion is fulfilled, as the standardized and well tested Advanced Encryption Standard (AES) in Output Feedback Mode (OFB) is used to encrypt the JPEG2000 codestream. On the other hand, the Transcodability criterion is not evaluated due to the fact that it is not relevant to this work. The following experiments and evaluations are based on the following images from the SCFace image database [46]: cam1_1, cam3_3, cam5_3, 001_frontal, (see Figure 7.1). In order to minimize the risk of outliers, the experimental results shown in this section are based on at least 1000 simulations. Hence, all the proposed embedding methods and parameterizations (Wavelet-Level and code-block-size) used to obtain the results are executed at least 1000 times (bitstream encryption, data embedding, data extraction and bitstream decryption).

This section is divided into two parts. The first part presents the evaluation of the experimental results obtained by the four proposed embedding methods, which are embedding the encryption specification into the JPEG2000 COM-segment, prior- of after-the JPEG2000 codestream, and embedding the encryption specification by replacing JPEG2000 image coefficients. Finally, the second part analyzes the results outlined in the first part, in conjunction with their suitability based on the criteria outlined in Section 3.2.

(a) cam1_1 **(b)** 1 RoI **(c)** 8 RoIs **(d)** cam3_3 **(e)** cam5_3 **(f)** 001_frontal

Figure 7.1: The figures above represent the data-set used to evaluate the proposed embedding methods. All surveillance images are based on the SCFace image database [46]. Figure 7.1a – image-size = 75×100 pixel and RoI = 36×72 pixel (top left corner = 26×12 pixel); Figure 7.1d – image-size = 168×224 pixel and RoI = 87×140 pixel (top left corner = 27×37 pixel); Figure 7.1e – image-size = 480×640 pixel and RoI = 90×158 pixel (top left corner = 252×392 pixel); Figure 7.1f – image-size = 768×1024 pixel and RoI = 579×804 pixel (top left corner = 104×36 pixel). Furthermore, the Figures 7.1b and Figure 7.1c show the image cam1_1 containing either one encrypted RoI or eight encrypted RoIs.

7.2.1 Embed Data into the JPEG2000 COM-Segment

Embedding the encryption specification into the JPEG2000 COM-segment offers the advantage of JPEG2000 format-compliance. However, in order to achieve format-compliance, the position of the COM-marker (0xFF64) within the JPEG2000 codestream must be detected and some corresponding COM-fields need to be altered according to the embedded encryption specification (for further details, see Subsection 6.2.1).

Table 7.1 shows the embedding overhead caused by embedding the encryption specification into the JPEG2000 COM-segment (indicated by *EMB*, which stands for active data embedding and represents the data overhead caused by embedding the encryption specification into the COM-segment). For comparability reasons, the overhead caused by storing all the start-, end-values and the encryption-counters is given as well (indicated by *No EMB*, which stands for no embedding). As evident from the results shown in Table 7.1, the overhead caused by the two signaling methods differs significantly. To be more precise, embedding the encryption specification into the COM-segment of image cam1_1 requires, on average, 3.010 % of the data-volume required by storing all the start-, end-values and the encryption-counters (4.879 % for image cam3_3, 9.660 % for image cam5_3 and 14.658 % for image 001_frontal). This is based on the fact that the proposed embedding methods embed the packet-IDs and their corresponding encryption counters only if the packet are encrypted more than once. Furthermore, as no packet start- or end-values of the encrypted packet is embedded into the codestream, this reduces the overhead as well. Finally, as evident from Table 7.1, using more Wavelet-Levels or a smaller code-block-size while encoding the JPEG2000 image causes an increased signaling overhead.

As evident from the experimental results, embedding the encryption specification into the COM-segment increases the computational demand required to encode a JPEG2000 image. This is due to the fact that the additional processing step is required to detect and alter the COM-segment, increasing the computing demand. Furthermore, decoding a JPEG2000 image used to embed the encryption specification requires more computing resources, as well, as the COM-segment needs to be detected and parsed prior to decrypting the encrypted image regions.

Table 7.2 shows the average time, in milliseconds, required to embed the encryption specification into the COM-segment and alter it as well. As evident with this, the time required to embed the encryption specification into the JPEG2000 codestream increases when smaller a code-block-size has been used while encoding the JPEG2000 image. This is due to the fact that smaller code-block-sizes cause an increased data-volume, which must be embedded into the JPEG2000 codestream. Hence, more computing resources are

WL	CBS	No EMB [byte]				EMB [byte]				Ratio [%]			
		1	2	3	4	1	2	3	4	1	2	3	4
0	4x4	144	276	312	312	8.64	30.27	49.69	65.53	6.00	10.97	15.93	21.00
	8x8	120	192	252	240	6.97	15.69	45.72	53.27	5.81	8.17	18.14	22.20
	16x16	108	180	192	216	6.89	13.88	33.22	48.77	6.38	7.71	17.30	22.58
	32x32	96	168	192	204	6.39	12.02	34.44	47.11	6.65	7.15	17.94	23.09
	64x64	96	168	192	204	6.26	12.36	34.50	47.03	6.52	7.35	17.97	23.05
1	4x4	276	492	516	612	9.49	32.31	58.29	89.45	3.44	6.57	11.30	14.62
	8x8	216	384	444	456	7.62	24.52	51.33	68.50	3.53	6.38	11.56	15.02
	16x16	180	288	408	408	7.21	12.93	44.55	62.24	4.01	4.49	10.92	15.25
	32x32	180	276	348	396	6.99	12.47	34.62	61.46	3.89	4.52	9.95	15.52
	64x64	168	276	324	396	6.48	12.11	26.86	62.34	3.86	4.39	8.29	15.74
2	4x4	564	684	756	900	13.13	33.21	77.20	140.80	2.33	4.85	10.21	15.64
	8x8	300	504	648	684	8.23	24.96	62.56	101.47	2.74	4.95	9.65	14.84
	16x16	276	360	612	624	7.73	13.71	59.13	89.75	2.80	3.81	9.66	14.38
	32x32	252	336	492	600	7.50	12.53	35.94	83.32	2.97	3.73	7.31	13.89
	64x64	252	324	492	576	7.10	12.94	34.60	80.19	2.82	3.99	7.03	13.92
3	4x4	720	864	1008	1200	13.59	33.03	81.03	159.68	1.89	3.82	8.04	13.31
	8x8	384	624	864	912	9.21	25.53	66.02	115.92	2.40	4.09	7.64	12.71
	16x16	324	456	660	840	7.81	14.00	61.01	102.28	2.41	3.07	9.24	12.18
	32x32	288	420	660	804	7.46	13.35	39.18	95.19	2.59	3.18	5.94	11.84
	64x64	288	408	660	768	7.49	12.90	39.23	89.65	2.60	3.16	5.94	11.67

Table 7.1: This table shows the embedding overhead caused by embedding the encryption specification into the JPEG2000 COM-segment. The abbreviation *EMB* stands for active embedding (in this case, into the COM-segment) and *No EMB* stands for no embedding (the encryption specification is stored in an additional file), which is stated for comparability reasons. The *No EMB* column represents the bytes required to store all the start-, end-values and the encryption counters. The *EMB* column represents the bytes required to store the encoded encryption specification into the COM-segment. *WL* stands for Wavelet-Level, *CBS* is the short form for code-block-size and *Ratio* shows the proportion between *No EMB* and *EMB* in %. All measurements are based on the average of 1000 simulations conducted on the following images from the SCFace image database [46]: cam1_1 **(1)**, cam3_3 **(2)**, cam5_3 **(3)** and 001_frontal **(4)**.

required to embed the encryption specification into the JPEG2000 codestream. Figure C.2 shows the results depicted in Table 7.2. Compared to Figure C.1, it is evident that the time required to embed the encryption specification depends on the data-volume which must be embedded into the JPEG2000 codestream.

However, as experimental results have shown, embedding the encryption specification into the COM-segment poses the risk of degrading the image quality. This is due to the fact that the EBCOT-algorithm (Embedded Block Coding with Optimal Truncation) inserts the truncation points differently when the main-header differs in length. Hence, embedding a different payload into the COM-segment while encoding the JPEG2000 image causes

WL	CBS	Embedding Time [ms]				WL	Embedding Time [ms]			
		1	**2**	**3**	**4**		**1**	**2**	**3**	**4**
	4x4	1.20	3.56	18.23	52.02		1.44	2.87	10.95	43.04
	8x8	1.12	2.80	11.95	33.35		1.34	2.34	7.89	30.07
0	16x16	1.02	2.38	9.75	28.94	2	1.22	2.10	5.78	23.45
	32x32	0.96	2.38	8.89	25.53		1.20	2.01	5.24	21.02
	64x64	0.93	2.30	8.68	24.18		1.17	1.95	5.44	20.62
	4x4	1.26	2.96	11.81	41.92		1.40	2.90	11.91	42.72
	8x8	1.28	2.38	7.99	29.11		1.34	2.43	7.69	29.80
1	16x16	1.05	2.11	6.34	22.55	3	1.36	2.16	6.26	22.94
	32x32	1.00	2.03	5.76	20.52		1.24	2.06	6.06	20.61
	64x64	0.99	1.94	5.59	19.57		1.32	1.97	5.73	20.52

Table 7.2: This table shows the time, in milliseconds, required to embed the encryption specification into the COM-segment. The abbreviation *CBS* stands for code-block-size and *WL* stands for Wavelet-Level. Both parameters are specified while encoding the JPEG2000 image. All measurements are based on the average of 1000 simulations conducted on the following images from the SCFace image database [46]: cam1_1 **(1)**, cam3_3 **(2)**, cam5_3 **(3)** and 001_frontal **(4)**.

a completely different JPEG2000 codestream. Therefore, modifying the COM-segment (embedding the encryption specification into the COM payload field, at codestream level) after encoding the JPEG2000 image might result in a degraded image quality. Therefore, this embedding approach might not be applicable for some applications, as the image quality might be degraded. However, as shown by experimental results, the issue of image quality degradation can be overcome by removing the embedded encryption specification prior to decoding the JPEG2000 image.

Table 7.3 shows the average SSIM, LSS, ESS and PSNR value caused by embedding the encryption specification into the JPEG2000 COM-segment of image cam1_1 (see Figure 7.1a). Therefore, the results are based on the decrypted and decoded sample image if the encryption specification is not removed prior to decoding the JPEG2000 image. Figure C.3 and Figure C.4 show the four image metrics described in Subsection 3.2.6, which are used to evaluate the image quality of the decrypted image. As evident from the results shown in these figures or in Table 7.3, the image quality is deteriorating when the code-block-size increases. This is due to the fact that the JPEG2000 encoder generates less data packets when the code-block-size increases. Therefore, the packets are larger and contain more encoded image coefficients, which could degrade the image quality more dramatically when decoded wrongly. The fluctuating PSNR value caused by COM-method is due to the EBCOT-algorithm (Embedded Block Coding with Optimal Truncation), which places the truncation points differently when the main-header differs in size. Hence, if the main

header differs in size, (compared to the size prior to embedding the encryption specification) the JPEG2000 decoder extracts the image data wrongly. However, as experiments have shown, a solution to solving the problem of degrading the image quality is to restore the original COM-segment prior to decoding the JPEG2000 image.

WL	CBS	SSIM	ESS	LSS	PSNR	WL	SSIM	ESS	LSS	PSNR
	4x4	1.000	1.000	1.000	∞		1.000	1.000	1.000	∞
	8x8	1.000	1.000	1.000	∞		0.939	0.962	0.887	65.843
0	16x16	0.982	0.983	0.927	27.151	2	1.000	1.000	1.000	∞
	32x32	0.999	0.998	0.996	∞		1.000	1.000	1.000	∞
	64x64	0.744	0.820	0.225	15.687		0.873	0.871	0.818	79.267
	4x4	1.000	1.000	1.000	∞		1.000	1.000	1.000	∞
	8x8	0.997	0.999	0.996	∞		1.000	1.000	1.000	∞
1	16x16	0.996	0.975	0.942	55.344	3	0.978	0.974	0.945	87.361
	32x32	0.849	0.890	0.839	49.262		0.697	0.808	0.506	39.999
	64x64	0.285	0.463	-0.871	11.489		0.627	0.651	0.446	42.831

Table 7.3: This table shows the four image quality metrics outlined in Subsection 3.2.6. The image quality scores are calculated after decrypting and decoding the JPEG2000 image cam1_1 (see Figure 7.1a into the PGM file-format, whereby the embedded encryption specification is not removed prior to decoding. The image quality metrics SSIM, ESS and LSS show the similarity between the original- and the image used for embedding (a similarity score of 1 indicates that the images are identical). The PSNR is given in dB and depicts the similarity between the original unmodified and the image used to embed the encryption specification (a higher dB score represents a better similarity, ∞ shows that the images are identical).

7.2.2 Embed Data prior to the JPEG2000 SOC-Marker

This subsection outlines the experimental results of embedding the encryption specification prior to the JPEG2000 SOC-marker (start of codestream). First, we need to point out that this approach of embedding data into the JPEG2000 codestream is not format-compliant. Hence, the image containing the embedded data might not be decodable by a standard compliant JPEG2000 decoder. This is due to the fact that Part 1 of the JPEG2000 standard defines that the JPEG2000 codestream starts by the SOC-marker (0xFF8E). To embed the encryption specification prior to the JPEG2000 codestream, the proposed embedding packet-structure must be created and prepended to the JPEG2000 encoded codestream (for further details about the proposed packet-structure and the embedding procedure, see Subsection 6.2.2).

Table 7.4 shows the embedding overhead caused by embedding the encryption specification prior to the JPEG2000 SOC-marker (indicated by *EMB*, which stands for active data embedding and represents the data overhead caused by embedding the encryption specification prior to the SOC-marker). For comparability reasons, the overhead caused

WL	CBS	No EMB [byte]				EMB [byte]				Ratio [%]			
		1	2	3	4	1	2	3	4	1	2	3	4
0	4x4	144	276	312	312	10.88	32.33	52.07	67.05	7.55	11.71	16.69	21.49
	8x8	120	192	252	240	9.06	17.76	47.42	55.40	7.55	9.25	18.82	23.08
	16x16	108	180	192	216	8.90	15.66	35.11	50.79	8.24	8.70	18.29	23.51
	32x32	96	168	192	204	8.26	13.85	36.41	49.14	8.60	8.24	18.96	24.09
	64x64	96	168	192	204	8.19	14.10	36.23	49.04	8.53	8.39	18.87	24.04
1	4x4	276	492	516	612	11.36	34.14	60.15	91.13	4.12	6.94	11.66	14.89
	8x8	216	384	444	456	9.77	26.34	53.57	70.72	4.52	6.86	12.07	15.51
	16x16	180	288	408	408	9.24	14.92	46.38	64.07	5.13	5.18	11.37	15.70
	32x32	180	276	348	396	8.85	14.70	36.56	63.42	4.92	5.33	10.51	16.02
	64x64	168	276	324	396	8.47	14.03	29.08	64.37	5.04	5.08	8.98	16.26
2	4x4	564	684	756	900	14.93	34.80	78.84	143.06	2.65	5.09	10.43	15.90
	8x8	300	504	648	684	10.37	27.20	64.33	103.65	3.46	5.40	9.93	15.15
	16x16	276	360	612	624	9.68	15.51	60.49	92.36	3.51	4.31	9.88	14.80
	32x32	252	336	492	600	9.26	14.40	37.58	84.81	3.68	4.29	7.64	14.14
	64x64	252	324	492	576	9.03	14.63	37.01	81.89	3.58	4.52	7.52	14.22
3	4x4	720	864	1008	1200	15.48	34.99	82.52	162.14	2.15	4.05	8.19	13.51
	8x8	384	624	864	912	11.06	27.44	67.99	117.92	2.88	4.40	7.87	12.93
	16x16	324	456	660	840	9.92	16.27	63.17	104.19	3.06	3.57	9.57	12.40
	32x32	288	420	660	804	9.47	15.14	40.95	97.78	3.29	3.61	6.20	12.16
	64x64	288	408	660	768	9.61	14.95	41.30	91.43	3.34	3.66	6.26	11.91

Table 7.4: This table shows the embedding overhead caused by embedding the encryption specification prior to the JPEG2000 SOC-marker (before JPEG2000 codestream). The abbreviation *EMB* stands for active embedding (in this case, prior to the SOC-marker) and *No EMB* stands for no embedding (the encryption specification is stored in an additional file), which is stated for comparability reasons. The *No EMB* column represents the bytes required to store all the start-, end-values and the encryption counters. The *EMB* column represents the bytes required to store the encoded encryption specification prior to the SOC-marker. *WL* stands for Wavelet-Level, *CBS* is the short form for code-block-size and *Ratio* shows the proportion between *No EMB* and *EMB* in %. All measurements are based on the average of 1000 simulations conducted on the following images from the SCFace image database [46]: cam1_1 **(1)**, cam3_3 **(2)**, cam5_3 **(3)** and 001_frontal **(4)**.

by storing all the start-, end-values and the encryption-counters is given as well (indicated by *EMB*, which stands for no embedding). As evident from the results shown in Table 7.4, the overhead caused by the two signaling methods differs significantly. To be more precise, embedding the encryption specification prior to the JPEG2000 codestream of image cam1_1 requires, on average, 3.857 % of the data-volume required by storing all the start-, end-values and the encryption-counters (5.380 % for image cam3_3, 10.039 % for image cam5_3 and 15.014 % for image 001_frontal). This is based on the fact that the proposed embedding methods embed the packet-IDs and their corresponding encryption counters only if the packet is encrypted more than once. Furthermore, the fact that no packet packet

start- or end-values of the encrypted packet is embedded into the codestream reduces the overhead as well.

As evident from the experimental results, embedding the encryption specification prior to the JPEG2000 SOC-marker increases the computational demand required to encode a JPEG2000 image. This is due to the fact that an additional step for prepending the encryption specification must be executed. Furthermore, decoding a JPEG2000 image which has been used to embed the encryption specification also requires more computing resources, as the prepended encryption specification must be detected and parsed prior to decrypting the encrypted image regions. Finally, the prepended data must be removed prior to decoding the image, otherwise a JPEG2000 standard compliant decoder might not be able to decode the image.

Table 7.5 shows the average time, in milliseconds, required to embed the encryption specification prior to the SOC-marker. However, as evident from the results shown below, it takes more time to embed the encryption specification for smaller code-block-sizes (CBS). This is due to the fact that smaller CBSs cause an increased encryption specification size. Hence, more data must be embedded into the JPEG2000 codestream. Figure C.6 shows the results depicted in the table below. Compared to Figure C.5 it is evident that the time required to embed the encryption specification depends on the data-volume which must be embedded into the JPEG2000 codestream.

WL	**CBS**	**Embedding Time [ms]**				**WL**	**Embedding Time [ms]**			
		1	**2**	**3**	**4**		**1**	**2**	**3**	**4**
	4x4	1.20	3.56	18.23	52.02		1.34	2.59	10.52	43.53
	8x8	1.12	2.80	11.95	33.35		1.08	2.06	7.16	29.25
0	16x16	1.02	2.38	9.75	28.94	2	0.85	1.88	5.70	23.48
	32x32	0.96	2.38	8.89	25.53		0.88	1.78	5.16	21.58
	64x64	0.93	2.30	8.68	24.18		0.97	1.75	4.94	20.23
	4x4	1.26	2.96	11.81	41.92		1.16	2.63	10.40	43.64
	8x8	1.28	2.38	7.99	29.11		0.98	2.07	7.72	27.77
1	16x16	1.05	2.11	6.34	22.55	3	0.98	1.88	5.68	23.23
	32x32	1.00	2.03	5.76	20.52		0.96	1.69	5.12	21.11
	64x64	0.99	1.94	5.59	19.57		1.05	1.79	5.38	20.19

Table 7.5: This table shows the time, in milliseconds, required to embed the encryption specification prior the JPEG2000 SOC-marker. The abbreviation *CBS* stands for code-block-size and *WL* stands for Wavelet-Level. Both parameters are specified while encoding the JPEG2000 image. All measurements are based on the average of 1000 simulations conducted on the following images from the SCFace image database [46]: cam1_1 **(1)**, cam3_3 **(2)**, cam5_3 **(3)** and 001_frontal **(4)**.

As experimental results have shown, if the embedded encryption specification is removed prior to decoding the JPEG2000 image, no image-quality degradation, or JPEG2000 format-compliance problems are detectable. This is based on the fact that the encoded JPEG2000 codestream is not modified while embedding the encryption specification. However, as the experimental results have shown, if the prepended data are not removed prior to decoding the image, a JPEG2000 standard compliant decoder cannot decode it.

7.2.3 Embed Data after the JPEG2000 EOC-Marker

This subsection outlines the experimental results of embedding the encryption specification after the JPEG2000 EOC-marker (end of codestream). First, we need to point out that this embedding approach is not JPEG2000 format-compliant, and also embeds the data prior to the codestream. Hence, the image containing the embedded data might not be decodable by a JPEG2000 standard compliant decoder. However, as experiments have shown, this embedding approach is more likely to be decodable by a standard compliant decoder (JJ2000 Version 5.1 and IrfanView Version 4.32 are able to decode the image).

Table 7.6 shows the embedding overhead caused by embedding the encryption specification after the JPEG2000 EOC-marker (indicated by *EMB*, which stands for active data embedding and represents the data overhead caused by embedding the encryption specification after the EOC-marker). For comparability reasons, the overhead caused by storing all the start-, end-values and the encryption-counters is given as well (indicated by *No EMB*, which stands for no embedding). The results in Table Table 7.6 illustrate, the overhead caused by the two signaling methods differs significantly. To be more precise, embedding the encryption specification after the EOC-marker of image cam1_1 requires, on average, 3.863 % of the data-volume required by storing all the start-, end-values and the encryption-counters (5.384 % for image cam3_3, 10.037 % for image cam5_3 and 15.015 % for image 001_frontal). This is based on the fact that the proposed embedding methods embed the packet-IDs and their corresponding encryption counters only if the packet is encrypted more than once. Furthermore, as no packet start- or end-values of the encrypted packet is embedded into the codestream, the overhead is also reduced.
As seen with the experimental results, embedding the encryption specification after the EOC-marker increases the computational demand required to encode a JPEG2000 image. This is due to the fact that an additional step for appending the encryption specification must be executed. Furthermore, decoding a JPEG2000 image also requires more computing resources, as the appended encryption specification must be detected and parsed prior to decrypting the encrypted image regions. Finally, the appended data must be removed

WL	CBS	No EMB [byte]				EMB [byte]				Ratio [%]			
		1	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
0	4x4	144	276	312	312	10.67	31.97	51.62	67.62	7.41	11.58	16.54	21.67
	8x8	120	192	252	240	9.16	17.76	47.34	55.39	7.64	9.25	18.79	23.08
	16x16	108	180	192	216	8.89	15.88	35.26	50.76	8.23	8.82	18.36	23.50
	32x32	96	168	192	204	8.41	14.00	36.59	48.83	8.76	8.33	19.05	23.94
	64x64	96	168	192	204	8.21	13.73	36.39	49.03	8.56	8.17	18.95	24.04
1	4x4	276	492	516	612	11.40	33.87	59.89	91.43	4.13	6.88	11.61	14.94
	8x8	216	384	444	456	9.84	26.34	53.53	71.10	4.56	6.86	12.06	15.59
	16x16	180	288	408	408	9.43	14.86	46.52	64.03	5.24	5.16	11.40	15.69
	32x32	180	276	348	396	9.14	14.44	36.73	63.54	5.08	5.23	10.55	16.04
	64x64	168	276	324	396	8.55	14.30	28.93	64.21	5.09	5.18	8.93	16.22
2	4x4	564	684	756	900	14.94	34.80	79.16	142.72	2.65	5.09	10.47	15.86
	8x8	300	504	648	684	10.34	26.99	64.44	103.35	3.45	5.35	9.94	15.11
	16x16	276	360	612	624	9.74	15.54	60.83	91.97	3.53	4.32	9.94	14.74
	32x32	252	336	492	600	9.29	14.80	37.61	85.41	3.68	4.40	7.64	14.24
	64x64	252	324	492	576	8.97	14.33	36.49	81.79	3.56	4.42	7.42	14.20
3	4x4	720	864	1008	1200	15.39	35.52	82.61	161.85	2.14	4.11	8.20	13.49
	8x8	384	624	864	912	10.94	27.41	68.30	118.19	2.85	4.39	7.90	12.96
	16x16	324	456	660	840	9.84	16.34	62.69	103.82	3.04	3.58	9.50	12.36
	32x32	288	420	660	804	9.41	15.73	40.98	97.73	3.27	3.75	6.21	12.15
	64x64	288	408	660	768	9.53	14.90	40.99	91.77	3.31	3.65	6.21	11.95

Table 7.6: This table shows the embedding overhead caused by embedding the encryption specification after the JPEG2000 EOC-marker (after JPEG2000 codestream). The abbreviation *EMB* stands for active embedding (in this case, after the EOC-marker) and *No EMB* stands for no embedding (the encryption specification is stored in an additional file), which is stated for comparability reasons. The *No EMB* column represents the bytes required to store all the start-, end-values and the encryption counters. The *EMB* column represents the bytes required to store the encoded encryption specification after the EOC-marker. *WL* stands for Wavelet-Level, *CBS* is the short form for code-block-size and *Ratio* shows the proportion between *No EMB* and *EMB* in %. All measurements are based on the average of 1000 simulations conducted on the following images from the SCFace image database [46]: cam1_1 **(1)**, cam3_3 **(2)**, cam5_3 **(3)** and 001_frontal **(4)**.

prior to decoding the image, otherwise a JPEG2000 standard compliant decoder might not be able to decode the image.

Table 7.7 shows the average time, in milliseconds, required to embed the encryption specification after the JPEG2000 EOC-marker. However, as the results below demonstrate, it takes more time to embed the encryption specification for smaller code-block-sizes. This is due to the fact that smaller CBSs cause an increased encryption specification size. Hence, more data must be embedded into the JPEG2000 codestream. Figure C.8 shows the results depicted in the Table 7.7. Compared to Figure C.7, it is evident that the time required to embed the encryption specification, depends on the data-volume which must be embedded into the JPEG2000 codestream.

WL	CBS	Embedding Time [ms]				WL	Embedding Time [ms]			
		1	2	3	4		1	2	3	4
	4x4	1.06	3.16	18.13	53.28		1.01	2.46	10.48	42.75
	8x8	0.91	2.31	11.87	35.65		1.01	2.03	7.48	28.77
0	16x16	0.87	2.02	9.65	26.83	2	0.81	1.70	5.69	23.49
	32x32	0.80	1.90	8.79	25.20		0.76	1.60	5.16	21.17
	64x64	0.78	1.88	8.66	25.93		0.87	1.67	4.97	20.95
	4x4	0.96	2.53	11.73	44.07		1.01	2.58	10.69	40.82
	8x8	0.84	1.97	7.90	30.09		0.92	1.98	7.29	28.45
1	16x16	0.79	1.71	6.22	23.87	3	0.90	1.82	6.20	22.25
	32x32	0.75	1.66	5.71	21.60		0.78	1.69	5.37	19.79
	64x64	0.77	1.58	5.52	20.75		0.87	1.69	5.23	19.04

Table 7.7: This table shows the time, in milliseconds, required to embed the encryption specification after the JPEG2000 EOC-marker. The abbreviation *CBS* stands for code-block-size and *WL* stands for Wavelet-Level. Both parameters are specified while encoding the JPEG2000 image. All measurements are based on the average of 1000 simulations conducted on the following images from the SCFace image database [46]: cam1_1 **(1)**, cam3_3 **(2)**, cam5_3 **(3)** and 001_frontal **(4)**.

7.2.4 Length-Preserving Data Embedding

This subsection outlines the experimental results of embedding the encryption specification into the JPEG2000 codestream by replacing JPEG2000 image coefficients. This method has the advantage, as mentioned in Subsection 6.2.4, of being JPEG2000 format-compliant and thereby preserving the original codestream length, which might be necessary for some applications. For further details on how the encryption specification is embedded into the JPEG2000 codestream, see Subsection 6.2.4.

Table 7.8 shows the number of image coefficients which have been replaced by the encryption specification (indicated by *EMB*, which stands for active data embedding and represents the JPEG2000 image coefficients, which are replaced by the encryption specification). As this method embeds the encryption specification by replacing image coefficients, no additional data overhead is caused. For comparability reasons, the overhead caused by storing all the start-, end-values and the encryption-counters is given as well (indicated by *No EMB*, which stands for no embedding). As demonstrated by the results in Table 7.8, the number of bytes required by the two signaling methods differs significantly. To be more precise, embedding the encryption specification into the JPEG2000 codestream by replacing image coefficients of image cam1_1 requires, on average, 3.860 % of the data-volume required by storing all the start-, end-values and the encryption-counters (5.373 % for image cam3_3, 10.052 % for image cam5_3 and 15.026 % for image 001_frontal). This is due to the fact that the proposed embedding packet structure stores only stores the

information required for decrypting the encrypted packets (no data which can be parsed from the JPEG2000 codestream itself is included anymore). Furthermore, the proposed embedding methods embed the packet-IDs and their corresponding encryption counters only if the packet is encrypted more than once.

WL	CBS	No EMB [byte]				EMB [byte]				Ratio [%]			
		1	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
0	4x4	144	276	312	312	10.56	32.13	51.67	67.25	7.34	11.64	16.56	21.55
	8x8	120	192	252	240	9.16	17.56	47.70	55.37	7.63	9.15	18.93	23.07
	16x16	108	180	192	216	8.65	15.97	35.07	50.82	8.01	8.87	18.27	23.53
	32x32	96	168	192	204	8.31	13.98	36.25	49.21	8.65	8.32	18.88	24.12
	64x64	96	168	192	204	8.26	13.85	36.33	48.81	8.60	8.24	18.92	23.93
1	4x4	276	492	516	612	11.19	34.32	60.50	91.35	4.06	6.98	11.72	14.93
	8x8	216	384	444	456	9.60	26.41	53.60	70.81	4.45	6.88	12.07	15.53
	16x16	180	288	408	408	9.29	14.57	46.44	64.10	5.16	5.06	11.38	15.71
	32x32	180	276	348	396	8.89	14.54	36.69	63.60	4.94	5.27	10.54	16.06
	64x64	168	276	324	396	8.57	14.02	28.97	64.60	5.10	5.08	8.94	16.31
2	4x4	564	684	756	900	15.29	34.85	79.16	142.86	2.71	5.10	10.47	15.87
	8x8	300	504	648	684	10.28	26.84	64.80	103.75	3.43	5.33	10.00	15.17
	16x16	276	360	612	624	9.67	15.74	60.64	91.79	3.50	4.37	9.91	14.71
	32x32	252	336	492	600	9.17	14.60	37.97	85.48	3.64	4.34	7.72	14.25
	64x64	252	324	492	576	9.13	14.61	36.41	81.98	3.62	4.51	7.40	14.23
3	4x4	720	864	1008	1200	15.70	35.11	83.08	162.29	2.18	4.06	8.24	13.52
	8x8	384	624	864	912	11.28	27.04	68.03	118.20	2.94	4.33	7.87	12.96
	16x16	324	456	660	840	9.92	16.29	63.06	103.93	3.06	3.57	9.55	12.37
	32x32	288	420	660	804	9.60	15.27	40.73	97.94	3.33	3.64	6.17	12.18
	64x64	288	408	660	768	9.50	14.98	41.36	91.62	3.30	3.67	6.27	11.93

Table 7.8: This table shows the experimental results for embedding encryption specification into the JPEG2000 codestream by replacing image coefficients. The abbreviation *EMB* stands for Embedding Data into the codestream and *No EMB* stands for no embedding has been applied, which is used to compare the proposed embedding method with the data-volume, which would be required for storing all the start-, end-values and the encryption-counters. *WL* stands for Wavelet-Level, *CBS* is the abbreviation for code-block-size and *Ratio* shows the proportion between *No EMB* and *EMB* in %. All measurements are based on the average of 1000 simulations conducted on the following images from the SCFace image database [46]: cam1_1 **(1)**, cam3_3 **(2)**, cam5_3 **(3)** and 001_frontal **(4)**.

As shown with the experimental results, embedding the encryption specification into JPEG2000 codestream by replacing image coefficients increases the computational demand required to encode a JPEG2000 image. This is due to the fact that an additional processing step required to embed the encryption specification into the JPEG2000 codestream must be executed. Furthermore, decoding a JPEG2000 image also requires more computing resources, as the embedded data must be detected and parsed prior to de-

crypting the encrypted image regions. This embedding method has the computational advantage of not removing the embedded data prior to decoding the JPEG2000 image. This is due to the fact that the image is still JPEG2000 format-compliant after embedding the encryption specification.

Table 7.9 shows the average time, in milliseconds, required for embedding the encryption specification into the JPEG2000 codestream by replacing JPEG2000 image coefficients. As shown here, the time required to embed the encryption specification into the JPEG2000 increases when a smaller code-block-size has been used while encoding the JPEG2000 image. This is due to the fact that smaller code-block-sizes cause an increased data-volume which must be embedded into the JPEG2000 codestream. Hence, more computing resources are required to embed the encryption specification into the JPEG2000 codestream. Figure C.10 shows the results depicted in Table 7.9. By comparing Figure C.10 with Figure C.9 it becomes evident that the time required to embed the encryption specification into the JPEG2000 codestream depends on the data-volume which must be embedded into the JPEG2000 codestream.

WL	**CBS**	**Embedding Time [ms]**				**WL**	**Embedding Time [ms]**			
		1	**2**	**3**	**4**		**1**	**2**	**3**	**4**
	4x4	1.13	3.16	19.43	54.73		1.08	2.45	11.58	43.71
	8x8	1.04	2.39	12.84	34.98		0.92	1.98	8.12	29.08
0	16x16	1.06	2.18	10.32	28.91	2	0.97	1.84	6.19	22.79
	32x32	0.99	2.04	9.53	27.00		0.89	1.72	5.58	20.98
	64x64	0.94	2.09	9.17	25.18		0.92	1.70	5.46	20.32
	4x4	1.08	2.49	12.61	45.14		1.04	2.45	11.58	41.04
	8x8	0.88	1.98	8.60	29.90		0.98	2.07	7.92	29.87
1	16x16	0.94	1.82	6.68	24.45	3	0.90	1.72	6.26	22.77
	32x32	0.88	1.76	6.09	22.08		0.89	1.58	5.65	20.23
	64x64	0.89	1.64	5.99	20.84		0.90	1.62	5.55	19.75

Table 7.9: This table shows the time, in milliseconds, required to embed the encryption specification into the JPEG2000 codestream, by replacing JPEG2000 image coefficients. The abbreviation *CBS* stands for code-block-size and *WL* stands for Wavelet-Level. Both parameters are specified while encoding the JPEG2000 image. All measurements are based on the average of 1000 simulations conducted on the following image from the SCFace image database [46]: cam1_1 (see Figure 7.1).

However, as the experimental results show, embedding the encryption specification into the JPEG2000 codestream by replacing image coefficients poses the risk of degrading the image quality. Furthermore, as proven by this study (see Subsection 7.3.5), the degree of image quality degradation depends on the data-volume which has been embedded into the JPEG2000 codestream. This is due to the fact that some JPEG2000 image coefficients are replaced by the encryption specification.

Table 7.10 shows the average SSIM, LSS, ESS and PSNR value caused by embedding the encryption specification into the JPEG2000 codestream by replacing image coefficients. All the experimental results shown in the table below are based on the results obtained by the decrypted and decoded sample image cam1_1 from the SCFace image database [46]. Due to the fact that all the other sample images used to evaluate the embedding methods (see Figure 7.1) show similar results, the following evaluation focuses on image cam1_1. Figure C.11 shows the four image metrics described in Subsection 3.2.6, which are used to evaluate the image quality of the decrypted image. As illustrated by the results in Figure C.11 or Table 7.10, the PSNR (peak-signal-to-noise-ratio) is deteriorates if more data is embedded into the JPEG2000 codestream. This is due to the fact that more JPEG2000 image coefficients are replaced by the encryption specification (see Figure C.12, which visualizes the PSNR values listed in Table 7.10 below). All the experiments conducted in this work showed no visible image degradations (at least for the human eye). This is due to the fact that no more than a few bytes have been replaced. However, as more image coefficients are replaced, the worse the image quality gets, as illustrated by the results below (smaller CBSs require more encryption specification, hence it degrades image quality to a greater degree).

WL	CBS	SSIM	ESS	LSS	PSNR	WL	SSIM	ESS	LSS	PSNR
	4x4	0.995	0.994	0.991	57.617		0.999	0.983	1.000	57.933
	8x8	0.996	0.991	0.992	58.464		1.000	0.987	1.000	66.356
0	16x16	0.999	0.986	0.999	70.786	2	1.000	0.983	1.000	70.622
	32x32	1.000	0.987	0.999	71.306		1.000	0.977	1.000	71.630
	64x64	1.000	0.988	1.000	73.837		1.000	0.994	1.000	73.178
	4x4	1.000	0.987	1.000	63.079		0.999	0.984	1.000	58.073
	8x8	1.000	0.986	1.000	68.447		1.000	0.985	1.000	65.490
1	16x16	1.000	0.983	1.000	70.564	3	1.000	0.984	1.000	70.378
	32x32	1.000	0.977	1.000	71.791		1.000	0.977	1.000	71.408
	64x64	1.000	0.997	1.000	73.364		1.000	0.992	1.000	72.769

Table 7.10: This table shows the four image quality metrics outlined in Subsection 3.2.6. The image quality scores are calculated after decrypting and decoding the JPEG2000 image into the PGM file-format, whereby embedding the encryption specification into the JPEG2000 codestream replaces some image coefficients. The image quality is degraded when compared with the original, unmodified image. The image quality metrics SSIM, ESS and LSS show the similarity between the original- and the image used for embedding (a similarity score of 1 indicates that the images are identical). The PSNR is given in dB and depicts the similarity between the original unmodified and the image used to embed the encryption specification (a higher dB score represents a better similarity, ∞ shows that the images are identical). All measurements are based on the average of 1000 simulations conducted on the following image from the SCFace image database [46]: cam1_1 (see Figure 7.1).

7.2.5 Encrypt Multiple RoIs per Image

As illustrated in Table 7.11 below, encrypting more than one RoI per JPEG2000 codestream causes no additional computational- or embedding overhead. This is due to the fact that this work utilizes the JPEG2000 *quality progression bitstream ordering*, as outlined in Chapter 6. This ordering-type offers the advantage of aligning all the JPEG2000 data packets belonging to an RoI or the data packets of multiple RoIs at the beginning of the JPEG2000 codestream. As shown in the table below, defining more than one RoI while encoding the JPEG2000 image (Figure 7.1a shows image cam1_1 from the SCFace image database [46], which has been used for multiple RoI encoding) has no impact on the length of the encryption specification, as all the RoI relevant data-packets are aligned at the beginning of the codestream. Table 7.11 shows embedding-overhead caused by embedding the encryption specification of up to eight different RoIs into the JPEG2000 COM-segment. As experimental results have shown, the crucial factor for determining the embedding-/decryption-time is the data-volume which must be embedded into the JPEG2000 codestream. However, this crucial factor is linked to the used Wavelet-Level, code-block-size and the size of the defined RoI, which all contribute to an increased encryption specification size. Hence, these parameters determine the time required to embed the data into the JPEG2000 image.

WL	CBS	COM-Segment – Embedding-Overhead							
		1 RoI	**2 RoI**	**3 RoI**	**4 RoI**	**5 RoI**	**6 RoI**	**7 RoI**	**8 RoI**
	8x8	6.292	6.247	6.316	6.244	5.248	5.266	5.137	5.293
	16x16	6.160	6.316	5.209	5.266	5.230	4.969	5.008	5.050
	32x32	6.247	6.244	5.266	5.293	4.939	5.050	5.023	5.002
	64x64	6.142	5.248	5.230	4.939	4.996	5.071	4.963	5.083
	4x4	6.316	5.266	4.969	5.050	5.071	5.038	10.372	10.498
	8x8	6.136	5.137	5.008	5.023	4.963	10.372	5.395	5.371
1	16x16	6.244	5.293	5.050	5.002	5.083	10.498	5.371	5.119
	32x32	5.209	4.969	5.011	5.038	10.264	5.362	4.945	4.999
	64x64	5.248	4.939	5.071	5.083	5.428	5.056	5.020	5.065
	4x4	5.236	4.948	5.014	10.087	5.362	5.107	4.978	10.327
	8x8	5.266	5.050	5.038	10.498	5.056	4.999	10.648	5.902
2	16x16	5.209	5.032	4.963	5.386	5.005	5.107	5.959	5.410
	32x32	5.137	5.023	10.372	5.371	5.020	10.648	5.245	5.476
	64x64	5.230	5.071	10.264	5.056	5.053	6.076	5.560	5.314

Table 7.11: This table shows the embedding overhead, in bytes, caused by embedding the encryption specification of up to eight RoIs into the JPEG2000 COM-segment. All measurements are based on the average of 1000 simulations conducted on image cam1_1 [46] (see Figure 7.1).

7.3 Comparison – Embedding Methods

Based on the results outlined in the previous section, the objective of this section is to compare the proposed embedding methods with each other. To do this, we will utilize the evaluation criteria described in Section 3.2, excluding the security - and transcodability aspects (as mentioned in the introduction to this chapter), which are not covered by this section due to the reasons previously outlined.

7.3.1 Format Compliance

As the experimental results show, not all proposed embedding methods fulfill the requirements of JPEG2000 format-compliance. Hence, not all of these methods can be used in conjunction with a standard compliant JPEG2000 Part 1 decoder, which might not be able to decode a non-format-compliant JPEG2000 image. Although embedding the encryption specification prior- or after the JPEG2000 codestream does not modify the encoded bitstream, these embedding methods do modify the JPEG2000 Part 1 codestream syntax (see Figure 2.8). This is based on the fact that after embedding the encryption specification, the codestream either has an invalid start- or end-value (JPEG2000 expects to start with the SOC-marker and ends with the EOC-marker). However, as experiments show, when embedding the encryption specification after the EOC-marker, the JJ2000 Version 5.1 coder and IrfanView Version 4.32 are still able to decode the image. However, the JJ2000 Version 5.1 coder shows a warning that the codestream ends incorrectly.
Embedding the encryption specification either into the JPEG2000 codestream by replacing image coefficients or into the JPEG2000 COM-segment are both format-compliant ways of embedding the encryption specification into the JPEG2000 codestream. This is based on the fact that no JPEG2000 marker is modified, added or deleted while embedding the encryption specification into the codestream. Furthermore, the JJ2000-decoder has not shown any format-compliance warnings while decoding the image, which would not be the case if the expected JPEG2000 syntax has been manipulated.

7.3.2 Embedding Overhead

As experimental results have shown, embedding the encryption specification into the COM-segment, prior- or after- the JPEG2000-codestream causes overhead. This is due to the fact, as outlined in Chapter 6, that the encryption specification must be added to the encoded and encrypted JPEG2000 codestream, which is necessary to successfully decrypt the encrypted bitstream parts. Furthermore, embedding the encryption specification into the JPEG2000 codestream by replacing image coefficients has the benefit of causing

no overhead, which might be mandatory for some applications. However, this method has the drawback of degrading the image-quality to a certain degree, as outlined in Subsection 6. As the results in Table 7.13 below illustrate, the proposed embedding methods (see Chapter 6) need less encryption specification than storing all the start-, end-values and the encryption-counters. Therefore, the following table gives an overview of the average (out of 1000 simulations) embedding overhead caused by the proposed embedding methods. The given results are based on image cam1_1 from the SCFace image database [46]. The other images from the data-set (see 7.1) show similar results, as presented in previous sections. As shown in the following table, the embedding overhead caused by the proposed embedding methods differs only slightly, except for embedding data into the COM-segment, which needs, on average, 2 bytes less overhead than the other methods. This is due to the fact that this method does not need to signal the start or the end of the encryption specification within the codestream.

WL	**CBS**	**No EMB [byte]**		**COM-Seg. [byte]**		**Before SOC [byte]**		**After EOC [byte]**		**Repl. Coef. [byte]**	
		1	**4**	**1**	**4**	**1**	**4**	**1**	**4**	**1**	**4**
0	4x4	144	312	8.64	65.53	10.88	67.05	10.67	67.62	10.56	67.25
	8x8	120	240	6.97	53.27	9.06	55.40	9.16	55.39	9.16	55.37
	1616	108	216	6.89	48.77	8.90	50.79	8.89	50.76	8.65	50.82
	32x32	96	204	6.39	47.11	8.26	49.14	8.41	48.83	8.31	49.21
	64x64	96	204	6.26	47.03	8.19	49.04	8.21	49.03	8.26	48.81
1	4x4	276	612	9.49	89.45	11.36	91.13	11.40	91.43	11.19	91.35
	8x8	216	456	7.62	68.50	9.77	70.72	9.84	71.10	9.60	70.81
	1616	180	408	7.21	62.24	9.24	64.07	9.43	64.03	9.29	64.10
	32x32	180	396	6.99	61.46	8.85	63.42	9.14	63.54	8.89	63.60
	64x64	168	396	6.48	62.34	8.47	64.37	8.55	64.21	8.57	64.60
2	4x4	564	900	13.13	140.80	14.93	143.06	14.94	142.72	15.29	142.86
	8x8	300	684	8.23	101.47	10.37	103.65	10.34	103.35	10.28	103.75
	1616	276	624	7.73	89.75	9.68	92.36	9.74	91.97	9.67	91.79
	32x32	252	600	7.50	83.32	9.26	84.81	9.29	85.41	9.17	85.48
	64x64	252	576	7.10	80.19	9.03	81.89	8.97	81.79	9.13	81.98
3	4x4	720	1200	13.59	159.68	15.48	162.14	15.39	161.85	15.70	162.29
	8x8	384	912	9.21	115.92	11.06	117.92	10.94	118.19	11.28	118.20
	1616	324	840	7.81	102.28	9.92	104.19	9.84	103.82	9.92	103.93
	32x32	288	804	7.46	95.19	9.47	97.78	9.41	97.73	9.60	97.94
	64x64	288	768	7.49	89.65	9.61	91.43	9.53	91.77	9.50	91.62

Table 7.13: This table compares the average capacity required to embed the encryption specification into the JPEG2000 codestream. Therefore, image cam1_1 **(1** ,see Figure 7.1) and image 001_frontal **(4** ,see Figure 7.1f) from the SCFace image database [46] have been used. Column three and four show the capacity, in bytes, required to store all the start-, end-values and the encryption-counters into an additional file (*No EMB*). The other columns show the capacity required to store the encryption specification into the JPEG2000 codestream. The abbreviation *WL* stands for Wavelet-Level and *CBS* stands for code-block-size.

Figure C.17 depicts the embedding overhead caused by the proposed embedding methods, as shown in Table 7.13. However, as the length-preserving data embedding does not cause any overhead, this figure shows the number of image coefficients which have been replaced by the encryption specification. As shown in Figure C.17, the capacity required to embed the encryption specification increases when the code-block-size decreases. As previously outlined, this is because a smaller code-block-size used while encoding the JPEG2000 image results in more JPEG2000 packets. Furthermore, encoding the image with more Wavelet-levels increases the data-volume required to store the encryption specification as well. Figure C.18 compares the proposed embedding methods with the embedding overhead which would be required to store all the start-, end-values and the encryption-counters.

7.3.3 Computational Demand

As outlined in previous sections decoding a JPEG2000 image which has been used to embed the encryption specification demands more computing resources. This is based on the fact that the embedded data must be extracted and parsed prior to decrypting the encrypted bitstream parts. Furthermore, as experimental results have shown, not only does the decoding require more computing resources; embedding the encryption specification into the JPEG20000 bitstream itself also requires more computing resources than would be needed for simply writing the data into an additional file.

As the results in Table 7.14 illustrate, the proposed embedding methods require more time to decrypt the decrypted bitstream parts than would be the case when reading the encryption specification from an additional file (indicated by *No EMB*, which stands for no data embedding). The given measurements are based on image cam1_1 from the SCFace image database [46]. The other images from the data-set show similar results and are therefore not explicitly mentioned in this section. The increased decryption time is due to the fact that some JPEG2000 codestream parts have been modified and not all the start-, end-values are stored in the JPEG2000 codestream. Hence, the missing information must be parsed from the JPEG2000 codestream, which requires more time. As shown in the following table, the time required by the proposed embedding methods differs only slightly, except for embedding data into the COM-segment, which requires, on average, one millisecond longer than embedding the encryption specification before or after the JPEG2000 codestream. Furthermore, embedding the data into the JPEG2000 codestream by replacing image coefficients requires, on average, half a millisecond less time than the other methods, as no data need to be removed prior to decoding the image. Figure C.19 depicts the decryption time required by the proposed embedding methods,

as shown in Table 7.14 below. As evident in this figure, the time required to decrypt the encrypted bitstream parts is lower when no embedding has been performed. This is due to the fact that the embedded encryption specification must be extracted and parsed prior to decrypting the encrypted bitstream parts, which is not the case when the whole encryption specification is stored in another file. Furthermore, this figure shows some peaks, which are caused by choosing a smaller code-block-size while encoding the JPEG2000 image, which then leads to more encrypted packet data.

Figure C.20 depicts the embedding time caused by the proposed embedding methods. As evident in this figure and previous analyzes (see the Tables 7.2, 7.5, 7.7 and 7.9) embedding data into the COM-Segment poses the highest computational demand since, in addition to embedding the encryption specification, some COM-fields must be altered, which is not the case for the other embedding methods.

WL	CBS	No EMB [ms]	COM-Seg. [ms]	Before SOC [ms]	After EOC [ms]	Repl. Coef. [ms]
0	4x4	3.40	8.83	8.87	8.87	7.92
	8x8	3.02	7.47	7.05	7.05	6.78
	1616	2.89	6.61	6.64	6.54	6.55
	32x32	2.71	5.83	5.79	5.77	5.62
	64x64	2.75	5.90	6.03	5.86	5.91
1	4x4	3.57	9.78	8.82	8.75	8.71
	8x8	3.29	9.30	7.61	7.58	6.77
	1616	3.00	7.05	5.95	6.00	5.85
	32x32	2.96	6.70	5.87	5.75	5.39
	64x64	2.83	6.36	5.45	5.71	5.30
2	4x4	4.54	15.78	13.97	12.95	13.08
	8x8	3.58	9.55	8.49	8.52	6.64
	1616	3.24	8.09	6.62	6.59	6.82
	32x32	3.18	7.91	6.73	6.29	6.23
	64x64	3.09	8.53	7.73	7.73	6.83
3	4x4	4.99	15.09	14.95	13.76	12.19
	8x8	3.79	10.79	8.92	9.09	8.16
	1616	3.51	8.78	7.84	7.09	6.14
	32x32	3.34	7.86	7.22	6.11	5.74
	64x64	3.25	8.18	7.62	7.01	5.74

Table 7.14: This table compares the average time, in milliseconds, required to decrypt the encrypted image regions. Therefore, image cam1_1 (see Figure 7.1) from the SCFace image database [46] has been used. Column three shows the time required to decrypt the JPEG2000 image when no data embedding has been applied (*No EMB*). The other columns shows the time required to extract, parse and decrypt the encrypted image parts (for further details see Chapter 6). The abbreviation *WL* stands for Wavelet-Level and *CBS* stands for code-block-size.

7.3.4 Image Quality

As outlined in the previous section, the image quality might be degraded by two of the proposed embedding methods. These methods are embedding the encryption specification into the JPEG2000 COM-segment (if the embedded data is not removed prior to decoding the JPEG2000 image) and the length-preserving method, which replaces JPEG2000 image coefficients by the encryption specification. As illustrated in Table 7.15 below, the length-preserving embedding method is in generally a better choice, as its image degradation is, on average, superior to the COM embedding method. The fluctuating PSNR value caused by COM-method is due to the EBCOT-algorithm, which places the truncation points differently when the main-header differs in length.

WL	CBS	COM SSIM	Repl. SSIM	COM ESS	Repl. ESS	COM LSS	Repl. LSS	COM PSNR	Repl. PSNR
0	4x4	1.000	0.995	1.000	0.994	1.000	0.991	99.924	57.617
	8x8	1.000	0.996	1.000	0.991	1.000	0.992	99.329	58.464
	16x16	0.982	0.999	0.983	0.986	0.927	0.999	27.151	70.786
	32x32	0.999	1.000	0.998	0.987	0.996	0.999	98.181	71.306
	64x64	0.744	1.000	0.820	0.988	0.225	1.000	15.687	73.837
1	4x4	1.000	1.000	1.000	0.987	1.000	1.000	100.000	63.079
	8x8	0.997	1.000	0.999	0.986	0.996	1.000	97.616	68.447
	16x16	0.996	1.000	0.975	0.983	0.942	1.000	55.344	70.564
	32x32	0.849	1.000	0.890	0.977	0.839	1.000	49.262	71.791
	64x64	0.285	1.000	0.463	0.997	-0.871	1.000	11.489	73.364
2	4x4	1.000	0.999	1.000	0.983	1.000	1.000	100.000	57.933
	8x8	0.939	1.000	0.962	0.987	0.887	1.000	65.843	66.356
	16x16	1.000	1.000	1.000	0.983	1.000	1.000	100.000	70.622
	32x32	1.000	1.000	1.000	0.977	1.000	1.000	100.000	71.630
	64x64	0.873	1.000	0.871	0.994	0.818	1.000	79.267	73.178
3	4x4	1.000	0.999	1.000	0.984	1.000	1.000	100.000	58.073
	8x8	1.000	1.000	1.000	0.985	1.000	1.000	100.000	65.490
	16x16	0.978	1.000	0.974	0.984	0.945	1.000	87.361	70.378
	32x32	0.697	1.000	0.808	0.977	0.506	1.000	39.999	71.408
	64x64	0.627	1.000	0.651	0.992	0.446	1.000	42.831	72.769

Table 7.15: This table shows the four image quality metrics outlined in Subsection 3.2.6. The image quality scores are calculated after decrypting and decoding the JPEG2000 image into the PGM file-format, whereby the encryption specification is either embedded into the COM-segment (indicated by *COM*) or into the JPEG2000 codestream by replacing JPEG2000 image coefficients (indicated by *Repl.*). The image quality metrics SSIM, ESS and LSS show the similarity between the original- and the image used for embedding (a similarity score of 1 indicates that the images are identical). The PSNR is given in dB and depicts the similarity between the original unmodified and the image used to embed the encryption specification (a higher dB score represents a better similarity, ∞ shows that the images are identical). All measurements are based on the average of 1000 simulations conducted on the following image from the SCFace image database: cam1_1 (see Figure 7.1).

7.3.5 Capacity Assessment – Length Preservation

As demonstrated by the experimental results, embedding the encryption specification into the JPEG2000 codestream by replacing image coefficients degrades the image quality. Therefore, this subsection evaluates the effect on the image quality caused by increasing the embedding capacity steadily. Table 7.16 shows the embedding capacity and the four image metrics, which are described in Section 3.2. All the measurements are based on the image cam1_1 (see Figure 7.1) (75x100 pixel) from the SCFace image database [46], which has been encoded with a code-block-size = 16x16 pixel and Wavelet decomposition level = 0. The encoded JPEG2000 image results in a file-size of 6718 bytes, which includes all the header-data, marker-segments and the encoded image coefficients stored in the packet bodies. Hence, subtracting the header-data and the marker-segments from the file-size results in 5921 bytes, which are used to store the encoded image coefficients. This size represents the maximum capacity that is feasible to embed into the JPEG2000 codestream without destroying JPEG2000 format-compliance. As evident in Figure C.13 and C.14, the image quality gets worse when the embedding capacity increases. Furthermore, Figure C.16 depicts the PSNR value caused by embedding up to 5921 bytes into the sample image, cam1_1 from the SCFace image database [46].

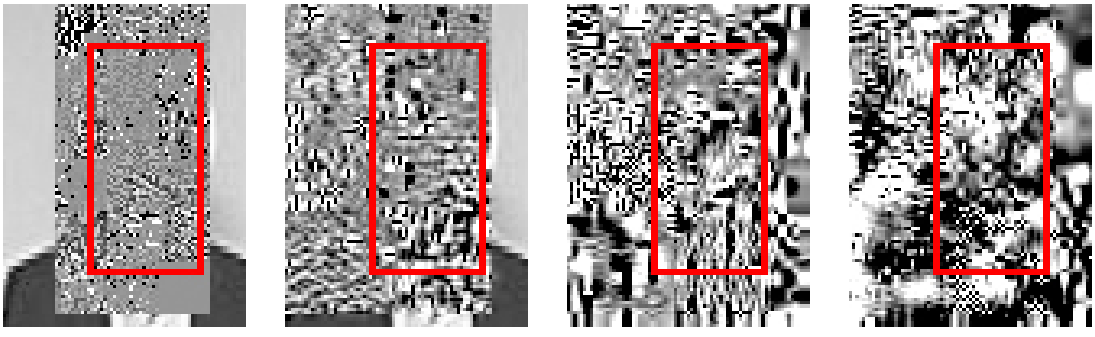
Embed. [byte]	**SSIM**	**LSS**	**ESS**	**PSNR**	**Embed. [byte]**	**SSIM**	**LSS**	**ESS**	**PSNR**
1	1	1	0.981	75.420	563	0.992	0.983	0.905	49.825
2	1	1	1	75.090	614	0.992	0.983	0.913	49.456
4	1	1	0.981	73.659	665	0.991	0.983	0.913	49.114
8	1	1	0.981	67.388	716	0.99	0.983	0.904	48.992
16	0.998	0.992	0.986	62.748	768	0.989	0.983	0.920	48.572
22	0.998	0.992	0.986	62.488	870	0.987	0.975	0.897	47.835
32	0.998	0.992	0.986	60.330	1024	0.984	0.975	0.862	47.224
71	0.998	0.992	0.986	58.011	1432	0.964	0.912	0.794	42.212
90	0.996	0.975	0.986	55.037	1636	0.952	0.878	0.778	40.477
128	0.996	0.983	0.986	54.998	2048	0.913	0.715	0.739	37.907
228	0.996	0.983	0.977	53.745	2864	0.739	0.202	0.611	28.900
256	0.995	0.983	0.971	52.911	3072	0.634	0.104	0.631	20.861
358	0.994	0.983	0.957	51.425	3480	0.351	0.050	0.571	11.779
409	0.994	0.983	0.949	51.069	3684	0.280	0.040	0.514	10.015
512	0.993	0.983	0.926	50.002	4096	0.283	0.030	0.522	10.215

Table 7.16: This table shows the four image quality metrics outlined in Subsection 3.2.6. The image quality scores are calculated after decrypting and decoding the JPEG2000 image into the PGM file-format, whereby the encryption specification is embedded into the JPEG2000 codestream by replacing image coefficients. All measurements are based on image cam1_1 [46], CBS of 16x16 pixel and a WL of 0.

7.4 Automated RoI Detection

This section presents and discusses the results of the proposed automated RoI detection methods. Therefore, the experimental results are based on the following images from the SCFace image database [46]: cam1_1 (see Figure 7.4a), cam3_3 (see Figure 7.4b), cam5_3 (see Figure 7.4c) and 001_frontal (see Figure 7.4d). All these images are encoded and evaluated for experimental purposes with varying code-block-sizes (CBS) and a constant Wavelet decomposition level (WL). As outlined by Uhl et al [39], the decision for encoding the input images with a constant Wavelet decomposition level is based on the fact that in case of small image dimensions or for higher Wavelet decomposition levels the spatial extent of a single code-block exceeds the RoIs or the image size, which would result in a decreased accuracy of detecting the encrypted JPEG2000 image region by the automated RoI detection methods proposed by this work.

In Figure 7.2, the impact of varying different encoded images on the scrambling accuracy of the JPEG2000 RoI method is demonstrated. Thereby, the decreasing covering accuracy of the predefined rectangular RoI (highlighted by red border) is clearly visible. The automated RoI detecting methods proposed by this work focus on Wavelet decomposition level 0, which offers the required property of an accurate RoI scrambling which does not exceed the code-block- nor its image-size. However, even if the image is encoded with WL-0, the encrypted image region exceeds the predefined rectangular RoI, which is highlighted by the red border. This circumstance is based on the fact that the JPEG2000 coder performs all the image processing steps on code-block basis (as outlined in Chapter 2). Hence, encrypting the majority of the image-coefficients belonging to one code-block destroys the visible reconcilability of the whole code-block.



(a) WL-0 **(b)** WL-1 **(c)** WL-2 **(d)** WL-3

Figure 7.2: The figures above show some RoI encryption examples (surveillance image cam1_1 from the SCFace image database [46]) with image size = 75×100 pixel and RoI = 36×72 pixel (RoI is highlighted by red border) with top left corner located at x = 26 and y = 12 pixel. All the images above are encoded with a code-block-size of 16×16 pixel and varying Wavelet-Levels. Figure 7.2a is encoded with a Wavelet-Level (WL) = 0, Figure 7.2b WL = 1, Figure 7.2c WL = 2 and Figure 7.2d WL = 3.

The proposed automated RoI detection methods have the following advantages over the proposed embedding methods (not all advantages apply to all embedding methods):

- **JPEG2000 format-compliance:** The syntactical and semantical requirements imposed by the JPEG2000 standard [9] are fulfilled
- **Losslessness:** The exact preservation of all (visible) picture data
- **Length-preserving:** It is guaranteed that the picture's file size does not change while performing the automated RoI detection

This is due to the fact that no data are embedded into the JPEG2000 codestream, nor are any image coefficients modified while performing the automated RoI detection.

7.4.1 Detect RoI by Entropy or Variance

This subsection outlines the experimental results of automatically detecting the encrypted picture region by its entropy or variance. Therefore, calculating the entropy or the variance is applied at three different input levels, as outlined in Chapter 6. The first method handles the RoI detection at PGM image data level, which is straight-forward, but as experimental results have shown (see Table 7.17), promises little success of detecting the encrypted image region correctly. As evident in Figure 7.3, the input image is further partitioned into smaller non-overlapping rectangular blocks, which are used to calculate the entropy/variance for each of these blocks. Each of these blocks is labeled in Figure 7.3 by a unique block-number, which is later used in Table 7.17 reference to certain image blocks. Although the human eye is capable of detecting encrypted picture region easily, the proposed automated RoI detection methods are not. Table 7.17 shows the experimental results of calculating the entropy/variance for each image block. All the measurements depicted in Table 7.17 represent the average of 1000 simulations, based on the circumstance that the input image is again re-encrypted after the entropy/variance has been calculated for the whole image. The following table highlights the unencrypted image blocks with a blue background color. However, as shown in the experimental results, detecting the encrypted image regions by either variance or entropy is not sufficient to accurately detect the encrypted image region automatically. On the one hand, this is based on the fact that some image blocks might have a high contrast, meaning that they are hardly

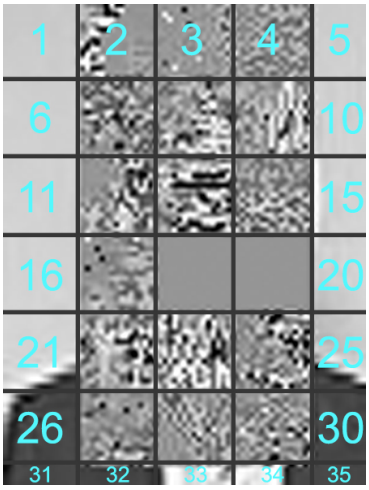


Figure 7.3: Partitioned PGM Input Image

distinguishable from encrypted image regions (see image block 21 or 25 in Figure 7.3). On the other hand, as experimental results have shown, detecting the encrypted image region by variance or entropy depends on the decryption itself. As illustrated in the image blocks 18 and 19 (see Figure 7.3), these two image blocks show little contrast. Hence, they do have a small variance- and entropy-value, which leads to the fact that detecting encrypted image blocks is even harder, as their variance-/entropy-values differ greatly from each other. Based on the experimental results, detecting the encrypted image region by variance or entropy does not work for PGM images, due to the above mentioned reasons.

Block	Variance	Entropy	Block	Variance	Entropy
1	5.968	3.248	19	2984.637	5.220
2	3017.622	5.265	20	68.288	4.638
3	2983.669	5.280	21	2226.279	4.921
4	3084.993	5.233	22	3049.670	5.241
5	6.338	3.035	23	3002.934	5.386
6	2.781	2.559	24	2986.943	5.319
7	2982.278	5.268	25	3449.446	6.074
8	2971.483	5.265	26	1529.972	4.900
9	3037.916	5.260	27	3154.024	5.231
10	15.057	3.708	28	2985.631	5.347
11	4.468	2.942	29	3026.227	5.259
12	3000.282	5.262	30	17.278	3.763
13	3055.838	5.295	31	79.069	3.233
14	3079.018	5.252	32	30.392	3.424
15	217.804	4.232	33	3110.082	5.282
16	3.761	2.971	34	8214.554	5.074
17	3112.561	5.251	35	1.208	1.961
18	3056.617	5.211			

Table 7.17: This table shows the experimental results of calculating the variance and the entropy based on a PGM image. Therefore, the source image has been encoded with a code-block-size of 16×16 pixel and a Wavelet-Level of 0. Image dimensions = 75×100 pixel. As illustrated in Figure 7.3 the input image is further partitioned into smaller non-overlapping rectangular picture regions, which are used to calculate the variance and entropy scores on image block basis. The highlighted cells above represent the unencrypted image parts. All measurements are based on the average of 1000 simulations conducted on the following image from the SCFace image database [46]: cam1_1.

Figure D.1 depicts the experimental results caused by calculating the entropy for each PGM-image block. Therefore, image cam1_1 from the SCFace image database [46] has been used. It has been encoded with a code-block-size of 16×16 pixel and a Wavelet decomposition level of 0. However, as evident in Figure D.1 it is not possible differ between an unencrypted image block or an encrypted image block. As outlined above, it is possible

that some unencrypted image blocks feature a high contrast and some encrypted image blocks feature a low contrast, which contradicts our assumptions that all encrypted image blocks should have increased entropy. Although calculating the variance shows similar results (see Figure D.2) it performs better than the entropy, as the difference between the monotonic gray background and the encrypted image parts is more clear. Nevertheless, when it comes to distinguishing between an encrypted image block and unencrypted images block with a high contrast (e.g., image block 21, 33, and 34), this methods shows the same problems as the entropy approach.

The second input level, which can be used to calculate the entropy or the variance, relies on extracting the image data from the JPEG2000 codestream. Therefore, the JPEG2000 codestream is parsed, as outlined in Subsection 6.3.1, to extract the JPEG2000 packet data, which contains the encoded and encrypted image coefficients. As shown in Table 7.18 below, calculating the entropy or the variance, based on data extracted from the JPEG2000 packets, do not promise automated encryption detection approaches. All the measurements shown in this table are based on the average of 1000 simulations and the JPEG2000 *quality progression bitstream ordering* has been applied to the codestream. Hence, the encrypted RoI image coefficients are aligned at the beginning of the JPEG2000

Packet-ID	Variance	Entropy	Packet-ID	Variance	Entropy
1	5380.761	5.074	13	5645.482	6.511
2	5273.485	4.793	14	5397.951	7.457
3	5374.257	5.712	15	5623.537	7.309
4	5173.898	4.380	16	4545.875	4.970
5	5280.161	4.494	17	4616.950	6.872
6	5401.586	5.580	18	5406.246	7.424
7	5467.336	6.316	19	5122.294	7.302
8	5423.892	6.740	20	5813.026	7.503
9	4937.823	6.977	21	5476.311	7.724
10	5754.693	6.104	22	5321.552	7.491
11	5371.121	7.564	23	5578.688	7.796
12	5533.204	6.893			

Table 7.18: This table shows the experimental results of calculating the variance and the entropy based on the data extracted from the JPEG2000 packet bodies. Therefore, the source image has been encoded with a code-block-size of 16×16 pixel and a Wavelet-Level of 0. Image dimensions = 75×100 pixel. Prior to calculating the variance and the entropy, the JPEG2000 packet body data must be extracted from the image codestream (JPEG2000 packets are encapsulated by SOP- and the EPH-marker). Hence, the packet data is used to calculate the variance and entropy scores for each packet. The highlighted cells represent the encrypted packets, which are located at the beginning of the JPEG2000 codestream (*quality progression bitstream ordering* has been applied while encoding the image). Packet number 9 represents a special case, as not the whole packet is encrypted, as is the case with the other highlighted packets.

codestream. To be more precise, the first 8 packets are completely taken by the encrypted RoI coefficients and packet number 9 is partly taken by the encrypted RoI coefficients. However, as shown in Table 7.18, encrypted entropy and the encrypted-variance are not distinguishable. This is based on the fact that the *Arithmetic entropy coding* (see Chapter 2) which is applied while encoding the image destroys any image characteristic, hence every packet looks the same. As experimental results show, detecting the encrypted JPEG2000 packets is not possible, due to the *Arithmetic entropy coding*. However, even if detecting the encrypted JPEG2000 packets would be possible, the issue of detecting the precise end of the encrypted codestream must be solved prior to accurately decrypting the encrypted image parts.
Figure D.3 depicts the experimental results caused by calculating the entropy of the JPEG2000 packet-data. As illustrated in the figure, it is not possible to differ between an encrypted (data-packets 1-8 are completely encrypted and packet 9 is partially encrypted, all the other data-packets are unencrypted) or any unencrypted data-packet. As depicted in Figure D.4, the same applies to computing the variance, as all the variance scores caused by the data-packets are quite similar. As mentioned above, this is due to the fact that the JPEG2000 encoder applies an entropy encoder prior to ordering and storing the JPEG2000 codestream. Therefore, as experiment results show, it is possible to distinguish between encrypted and unencrypted JPEG2000 data-packets.

Finally, the third input level which can be used to calculate the entropy or the variance relies on extracting the image data from the JPEG2000 inverse Wavelet transformation. Therefore, the encoded and encrypted JPEG2000 image is decoded until the inverse Wavelet transformation has been applied. Hence, the resulting inverse Wavelet coefficients are partitioned into smaller non-overlapping rectangular blocks, which are used to calculate the variance/entropy. For best results, as experiments have shown, the same code-block-size used to encode the image should be used to partition the image. Table 7.19 presents the experimental results caused by calculating the variance/entropy based on the values extracted from the inverse Wavelet-transformation. As shown in Table 7.19, the distinction between the encrypted- and unencrypted image blocks seems easier than with the PGM-method, but due to image blocks such as block 34, an accurate distinction is not possible. Although this input level is more likely to achieve higher encryption-detection accuracy, as would be the case with the other two input levels, it is obvious that more research must be done in this field in order to detect the encrypted image region accurately. However, as this method promises no success in detecting the encrypted bitstream parts, we chose not to focus on any further about detecting the encrypted image.

Block	Variance	Entropy	Block	Variance	Entropy
1	5.968	3.248	19	7760.816	5.508
2	8123.017	5.409	20	68.288	4.638
3	8907.141	5.394	21	2226.279	4.921
4	8159.228	5.499	22	8034.414	5.517
5	6.338	3.035	23	7937.078	5.464
6	2.781	2.559	24	8342.943	5.472
7	8614.466	5.600	25	3449.446	6.074
8	8413.303	5.555	26	1529.972	4.900
9	8482.343	5.561	27	8345.871	5.524
10	15.057	3.708	28	8635.804	5.541
11	4.468	2.942	29	7934.832	5.528
12	8267.022	5.534	30	17.278	3.763
13	8480.741	5.519	31	79.069	3.233
14	8454.742	5.499	32	30.392	3.424
15	217.804	4.232	33	3110.082	5.282
16	3.761	2.971	34	8214.554	5.074
17	8468.571	5.500	35	1.208	1.961
18	8041.758	5.497			

Table 7.19: This table shows the experimental results of calculating the variance and the entropy based on the data extracted from the JPEG2000 inverse Wavelet transformation. Therefore, image cam1_1 from the SCFace image database [46] has been used as source image to calculate the variance/entropy (encoded with a code-block-size of 16×16 pixel and a Wavelet-level of 0). The data extracted from the inverse Wavelet transformation is further partitioned into smaller non-overlapping rectangular blocks (see Figure 7.3). These blocks are used to calculate the variance and entropy on image block basis. The highlighted cells above represent the unencrypted image parts. All measurements are based on the average of 1000 simulations.

Figure D.5 depicts the experimental results caused by calculating the entropy for each image block. Therefore, image cam1_1 from the SCFace image database [46] has been used, which has been encoded with a code-block-size of 16×16 pixel and a Wavelet decomposition level of 0. Nevertheless, while the results are superior to all the other entropy calculation approaches, they feature the same drawback when it comes to distinguishing between an encrypted image block and an image block containing high contrast (e.g., image blocks 21, 25, 33 or 35). However, as demonstrated in Figure D.6, calculating the variance based on the data extracted from the inverse Wavelet transformed promises a higher chance of being able to distinguish between an encrypted- and an unencrypted image block. Nevertheless, even if detecting the encrypted image blocks would be sufficient, it would not be possible to get the encrypted bitstream from only detecting the encrypted image-blocks, as the encrypted RoI size does not match with the used code-block-size.

7.4.2 Detect RoI by Thresholding

Thresholding is the simplest and, as experimental results have shown, the most promising approach for automatically detecting the encrypted RoI. As outlined in Subsection 6.3.4, Thresholding is exclusively performed on the data extracted from JPGE2000s inverse Wavelet transformation. This is based on the fact that the other input sources (PGM-file, or JPEG2000 packet data) are not applicable for Thresholding, as they either consist only of valid image values (PGM-file) or are "randomly" distributed (due to the JPEG2000 entropy coder), and therefore offer no useable threshold. For calculating the threshold, the extracted inverse Wavelet coefficients, which can be represented as an image with width and height, are further partitioned into non-overlapping rectangular blocks, which are used to determine whether the block is encrypted or not. Hence, an unencrypted block consists solely of luminance values ranging from 0-255 (8 bit images). Encrypted image blocks might contain values exceeding these limits and therefore can be easily detected.
Table 7.20 shows the time required to determine the encrypted image region. As shown in Table 7.20, the time required to automatically detect the encrypted image region decreases when a larger code-block-size has been used while encoding the JPEG2000 image. One of the reasons for this is the fact that the same block-size is later used to partition the extracted inverse Wavelet coefficients into smaller non-overlapping rectangular blocks. Hence, a larger code-block-size leads to less blocks which need to be checked regarding whether they contain any image value exceeding the threshold or not. The results from Table 7.20 are illustrated in Figure D.7.

Image	**CBS**	**Time [ms]**	**Image**	**CBS**	**Time [ms]**
	4x4	19.122		4x4	706.752
	8x8	8.935		8x8	309.895
1	16x16	5.359	3	16x16	165.859
	32x32	4.747		32x32	116.396
	64x64	4.120		64x64	102.939
	4x4	98.318		4x4	1955.464
	8x8	42.570		8x8	803.133
2	16x16	23.619	4	16x16	422.808
	32x32	17.724		32x32	328.791
	64x64	16.570		64x64	310.280

Table 7.20: This table shows the time, in milliseconds, required to detect the encrypted image parts by the automated RoI detection method Thresholding. Therefore, the following images from the SCFace image database [46] have been used (image dimensions in pixel and reference number are given in brackets): cam1_1 (75×100, **1**), cam3_3 (168×224, **2**), cam5_3 (480×640, **3**), 001_frontal (768×1024, **4**).

Table 7.21 shows the error caused by determining the encrypted image region by the automated RoI detection method Thresholding. Therefore, the error caused by detecting each border of the encrypted region correctly is given. For example, the column **left X** indicates the number of wrongly determined left x-coordinate of the encrypted image region (out of 1000 simulations). Furthermore, the experimental results are based on four different images from the SCFace image database [46] (for further details see table caption). As demonstrated in the experimental results, the code-block-sizes 4×4 pixel or 64×64 pixel cause the highest classification error rate. Furthermore, as shown in the results below, most errors are caused by determining the y-coordinate. One reason for this is that the encrypted image region has more height than width. Hence, less data is available to determine the height of the encrypted image region correctly. As shown in the results below, the best RoI detection results are achievable by a code-block-size of 16×16 pixel.

Image	CBS	left X	upper Y	right X	lower Y	Error Abs.	Error Rate [%]
1	4x4	1	40	2	30	70	7.0
	8x8	0	7	1	16	23	2.3
	16x16	0	6	0	6	12	1.2
	32x32	2	8	0	6	16	1.6
	64x64	4	79	4	71	146	14.6
2	4x4	0	1	0	0	1	0.1
	8x8	0	0	0	7	7	0.7
	16x16	0	0	0	0	0	0.0
	32x32	0	0	0	0	0	0.0
	64x64	0	4	0	3	7	0.7
3	4x4	0	0	0	0	0	0.0
	8x8	0	0	0	0	0	0.0
	16x16	0	0	0	0	0	0.0
	32x32	0	0	0	0	0	0.0
	64x64	0	0	0	0	0	0.0
4	4x4	0	0	0	0	0	0.0
	8x8	0	0	0	0	0	0.0
	16x16	0	0	0	0	0	0.0
	32x32	0	0	0	0	0	0.0
	64x64	0	0	0	0	0	0.0

Table 7.21: This table shows the error caused by the automated RoI detection method Thresholding. The following images from the SCFace image database [46] have been used (image size in pixel and reference number are given in brackets): cam1_1 (75×100, **1**), cam3_3 (168×224, **2**), cam5_3 (480×640, **3**) and 001_frontal (768×1024, **4**). The results given in the columns: **left X**, **upper Y**, **right X** and **lower Y** represent the wrongly determined borders of the encrypted RoI (e.g., **left X** represents the upper left x-coordinate and **upper Y** the upper left Y-coordinate of the encrypted RoI). **Error Abs.** represents the wrongly detected RoI dimensions (out of 1000 simulations) and **Error Rate** shows the error in percent.

Figure D.8 depicts the error rate in percent, given in Table 7.21. As illustrated in the figure and the table, the code-block-sizes 4×4 pixel and 64×64 pixel cause the worst encryption detection accuracy. Furthermore, as demonstrated, smaller image-sizes pose the risk of an increased error rate. One reason therefore is due to the smaller encrypted RoI size, which lowers the chance of detecting an invalid (exceeding threshold) JPEG2000 image coefficient correctly. Another reason, as illustrated in Figure D.9, is that the error caused by detecting the y-coordinate correctly is increased for smaller images. This is based on the fact that the encrypted RoI has more height than width, hence less encrypted data is available for detecting the y-coordinate correctly. The increased error rate caused by detecting the y-coordinate correctly is highlighted in Figure D.10 and Figure D.11 that focus, due to the highest error rate, on a code-block-size of 4×4 pixel and 64×64 pixel.

7.4.3 Use Edge Detector to Detect RoI

This subsection presents the experimental results of automatically detecting the encrypted image region by its edge information. Therefore, three different edge detection methods have been evaluated for their suitability, when it comes to detecting the encrypted image region automatically. All the proposed methods have the benefit of not manipulating any JPEG2000 image coefficients nor embedding any information into the JPEG2000 codestream. Hence, the JPEG2000 codestream, after performing the encryption detection, is still format-compliant, no image quality degradation is measurable and the codestream length is preserved. Figure 7.4 depicts the data-set used to automatically detect the encrypted image region. All the images are based on the surveillance images from the SCFace image database [46].

(a) cam1_1 **(b)** cam3_3 **(c)** cam5_3 **(d)** 001_frontal

Figure 7.4: The figures above represent the data-set used by the automated edge detection methods. All surveillance images are based on the SCFace image database [46]. Figure 7.4a - image-size = 75×100 pixel and RoI = 36×72 pixel (top left corner = 26×12 pixel); Figure 7.4b - image-size = 168×224 pixel and RoI = 87×140 pixel (top left corner = 27×37 pixel); Figure 7.4c - image-size = 480×640 pixel and RoI = 90×158 pixel (top left corner = 252×392 pixel); Figure 7.4d - image-size = 768×1024 pixel and RoI = 579×804 pixel (top left corner = 104×36 pixel).

Sobel Edge Detection

As shown in Table 7.22, the encryption detection accuracy achieved by the Sobel edge detector is very low. Furthermore, the error rate is frequently at 100 %, which indicates that at least one border of the detected image region has been detected wrongly. This is based on the fact that the edges detected by the Sobel edge detector are not as precise as required to detect the borders of the encrypted image region more accurately, as shown in Figure 7.5 and outlined in Subsection 5.3.1. In other wordsthe edges detected by the Sobel edge-detector are blurred, as pixels located close to the edge are also linked to the edge. Hence, the detected edge cannot be represented by a thin line, but rather a thicker and more imprecise line instead. The proposed encryption detection method cannot accurately detect where the border of the encrypted image is located. Despite the increased accuracy which has been achieved for the whole data-set (see Table 7.22), when encoding the image with a code-block-size of 32×32 pixel, it is clear that perfect detection is not possible, albeit necessary for correct image decryption.

Figure 7.5 depicts the results of applying the Sobel edge detector to image cam5_3 from the SCFace image database [46]. As illustrated in Figure 7.5, the detected image edges are quite blurry, which makes a proper detection of the encrypted image region more complex. This is due to the fact that the detected edges are not exactly at the code-block borders, but all pixels around the actual edge are detected to belong to the edge. Another reason is that the edges running vertically or horizontally pose a major problem for detecting the encrypted picture region correctly. This is especially the case when a smaller code-block-size (e.g., 4×4 pixel) has been used to encode the image. Hence, if another edge runs parallel to the actual encryption border, the correct detection of the edge surrounding the encrypted image region is unlikely.

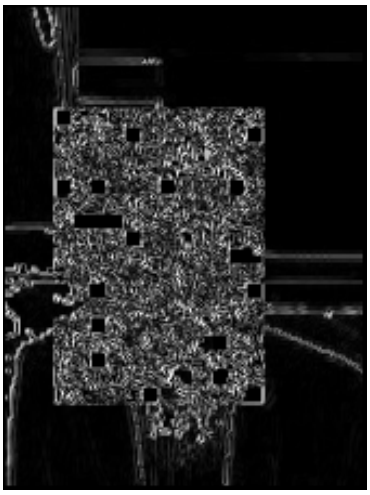


Figure 7.5: Detected Edges using Sobel Edge Detector - Image cam5_3 (CBS = 4×4 pixel, WL = 0)[46]

As illustrated in Figure D.12, which depicts the error caused by the Sobel edge detector logarithmically, the accuracy for detecting the encrypted image region correctly is very low. Furthermore, as outlined above, choosing a code-block-size of 4×4 pixel results in the worst classification performance.

Image	CBS	left X	upper Y	right X	lower Y	Error Abs.	Error Rate [%]
	4x4	75	317	25	1000	1000	100.0
	8x8	1	122	1	2	126	12.6
1	16x16	0	0	0	29	29	2.9
	32x32	0	0	0	10	10	1.0
	64x64	0	2	0	1000	1000	100.0
	4x4	129	250	111	94	433	43.3
	8x8	0	69	11	6	83	8.3
2	16x16	0	19	1	2	22	2.2
	32x32	0	0	0	0	0	0.0
	64x64	0	0	0	2	2	0.2
	4x4	1000	8	0	3	1000	100.0
	8x8	1000	2	0	0	1000	100.0
3	16x16	0	1	0	0	1	0.1
	32x32	0	13	0	1	14	1.4
	64x64	0	5	0	5	10	1.0
	4x4	0	0	0	1000	1000	100.0
	8x8	0	0	0	0	0	0.0
4	16x16	0	0	0	0	0	0.0
	32x32	0	0	0	0	0	0.0
	64x64	0	0	0	0	0	0.0

Table 7.22: This table shows the classification error caused by the Sobel edge detector. The following images from the SCFace image database [46] have been used (image size in pixel and reference number are given in brackets): cam1_1 (75×100, **1**), cam3_3 (168×224, **2**), cam5_3 (480×640, **3**) and 001_frontal (768×1024, **4**). The results given in the columns: **left X**, **upper Y**, **right X** and **lower Y** represent the wrongly determined borders of the encrypted RoI (e.g., **left X** represents the upper left x-coordinate and **upper Y** the upper left Y-coordinate of the encrypted RoI). **Error Abs.** represents the wrongly detected RoI dimensions (out of 1000 simulations) and **Error Rate** shows the error in percent.

Canny Edge Detection

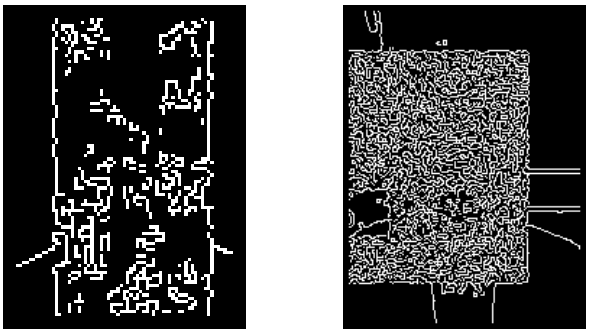
The second edge detection method proposed by this work relies on the Canny edge detector. This method, as shown in Table 7.23, performs as badly as the Sobel edge detection method. Although the Canny edge detector is much better at detecting encrypted image regions not bordering to the image borders, it fails when detecting the encrypted image region bordering to the image borders. Furthermore, the error rate is frequently at 100 %, which indicates that at least one border of the detected image region has been detected wrongly. This is based on the fact that the proposed Canny edge detector cannot detect edges located at the image borders, as shown in Figure 7.6b. As illustrated in Table 7.23, this results in the fact that all images containing an encrypted image region bordering the image border are not correctly detected. Despite the increased accuracy which has

been achieved for the whole data-set (see Table 7.23), when the encrypted image-region does not border the image borders, it is clear that perfect detection is not possible, albeit necessary for correct image decryption.

Image	CBS	left X	upper Y	right X	lower Y	Error Abs.	Error Rate [%]
	4x4	0	80	1	0	81	8.1
	8x8	0	2	0	0	2	0.2
1	16x16	0	1000	0	123	1000	100.0
	32x32	1000	1000	0	131	1000	100.0
	64x64	1000	1000	0	1000	1000	100.0
	4x4	7	21	17	21	60	6.0
	8x8	0	0	0	6	6	0.6
2	16x16	0	0	0	1	1	0.1
	32x32	1000	0	0	0	1000	100.0
	64x64	1000	1000	0	0	1000	100.0
	4x4	0	1000	0	0	1000	100.0
	8x8	0	0	0	0	0	0.0
3	16x16	0	0	0	0	0	0.0
	32x32	0	0	0	0	0	0.0
	64x64	0	0	0	0	0	0.0
	4x4	0	0	0	0	0	0.0
	8x8	0	0	0	0	0	0.0
4	16x16	0	0	0	0	0	0.0
	32x32	0	0	0	0	0	0.0
	64x64	0	1000	0	0	1000	100.0

Table 7.23: This table shows the classification error caused by the Canny edge detector. The following images from the SCFace image database [46] have been used (image size in pixel and reference number are given in brackets): cam1_1 (75×100, **1**), cam3_3 (168×224, **2**), cam5_3 (480×640, **3**) and 001_frontal (768×1024, **4**). The results given in the columns: **left X**, **upper Y**, **right X** and **lower Y** represent the wrongly determined borders of the encrypted RoI (e.g., **left X** represents the upper left x-coordinate and **upper Y** the upper left Y-coordinate of the encrypted RoI). **Error Abs.** represents the wrongly detected RoI dimensions (out of 1000 simulations) and **Error Rate** shows the error in percent.

As illustrated in Figure D.13, which depicts the error caused by the proposed Canny edge detector logarithmically, the accuracy for detecting the encrypted image regions correctly is very low. Furthermore, images containing an encrypted image region bordering the image border result in the worst classification performance. This is due to the simple fact that the proposed Canny edge detector cannot detect edges located at the image borders.



(a) cam1_1 **(b)** cam5_3

Figure 7.6: The sample figures above show the edges detected by the Canny Edge Detector - Figure 7.6a depicts image cam1_1 (image-size = 75×100 pixel, encrypted image region = 48×96 (top left corner: x = 16, y = 0 pixel), WL = 0, CBS = 16×16 pixel) and Figure 7.6b depicts image cam5_3 (image-size = 480×640 pixel, encrypted image region = 272×416 pixel (top left corner: x = 80, y = 144 pixel), WL = 0, CBS = 16×16 pixel) from the SCFace image database [46].

Proposed Edge Detection

As the edge detection methods described thus far lack satisfactory encryption detection accuracy, we have proposed another edge detection method. In order to achieve a better encryption detection performance, we extracted the data required to determine the edges surrounding the encrypted image region from the JPEG2000 inverse Wavelet transformation. This decision is based on the fact that this resource offers the advantage of getting higher contrast values between the unencrypted- (within range of 0-255, as a 8 bit image has been used) and the encrypted- image regions (might exceed valid image range). Next, the code-block-size used to encode the JPEG2000 image is extracted from the JPEG2000 decoder and used to determine the possible encryption borders. Finally, the contrast between one code-block and its adjoining code-block is calculated row by row and column by column. For example, if the code-block-size is 16×16 pixel, the contrast between image pixel 16 and 17, 32 and 33 until the end of the column is reached, is determined. Then the next column is checked and so on, until the end of the image is reached. The same is performed for the rows. Thereafter, each code-block border contains information regarding whether it borders the encrypted image region or not. Hence, the encrypted image region can be determined.

Table 7.24 presents the experimental results caused by the proposed edge detection method. As shown in Table 7.24, our edge-detection method is superior to the other evaluated automated edge detection methods. Although the human eye is capable of detecting encrypted image region easily, automatically detecting the edges belonging to an encrypted image region is quite complex. In particular, edges running vertically or horizontally pose a major problem for detecting the encrypted image region correctly. For instance, image cam5_3

from the SCFace image database [46], which contains many vertically and horizontally aligned edges, poses great difficulties in detecting the edges correctly. This circumstance, as outlined in the previous edge detection method, results in the fact that code-blocks bordering such "natural" edges are exposed to a higher risk of determining the edges of the encrypted image region incorrectly.

Image	CBS	left X	upper Y	right X	lower Y	Error Abs.	Error Rate [%]
1	4x4	0	0	0	0	0	0.0
	8x8	0	0	0	0	0	0.0
	16x16	0	66	0	0	66	6.6
	32x32	4	20	0	0	24	2.4
	64x64	15	52	0	26	85	8.5
2	4x4	0	0	2	25	25	2.5
	8x8	0	0	0	5	5	0.5
	16x16	0	0	0	1	1	0.1
	32x32	4	0	0	0	4	0.4
	64x64	0	37	0	0	37	3.7
3	4x4	0	1000	0	0	1000	100.0
	8x8	0	1000	0	0	1000	100.0
	16x16	0	1000	0	0	1000	100.0
	32x32	0	0	0	0	0	0.0
	64x64	0	0	0	0	0	0.0
4	4x4	0	0	0	0	0	0.0
	8x8	0	0	0	0	0	0.0
	16x16	0	0	0	0	0	0.0
	32x32	0	0	0	0	0	0.0
	64x64	0	0	0	0	0	0.0

Table 7.24: This table shows the error caused by the automated RoI detection method we have proposed. The following images from the SCFace image database [46] have been used (image size in pixel and reference number are given in brackets): cam1_1 (75×100, **1**), cam3_3 (168×224, **2**), cam5_3 (480×640, **3**) and 001_frontal (768×1024, **4**). The results given in the columns: **left X**, **upper Y**, **right X** and **lower Y** represent the wrongly determined borders of the encrypted RoI (e.g., **left X** represents the upper left x-coordinate and **upper Y** the upper left Y-coordinate of the encrypted RoI). **Error Abs.** represents the wrongly detected RoI dimensions (out of 1000 simulations) and **Error Rate** shows the error in percent.

Figure D.15 depicts the evaluated edge detection approaches. As illustrated in this figure, the method we have proposed is superior to all the other evaluated edge detection methods. Although the proposed edge detection method achieves a high accuracy, it is clear that perfect detection is not possible, albeit necessary for correct decryption. Therefore, the automated encryption detection methods have failed the real world feasibility test.

Chapter 8

Summary & Conclusion

We have proposed and evaluated several methods to signal multiple RoIs and to automatically detect a single RoI in the JPEG2000 codestream. Therefore, our implementation of the embedding and automated RoI detection methods builds up on Stubhann's implementation of a JPEG2000 RoI bitstream encryption [5], which follows an approach proposed by Hämmerle-Uhl et al. [19].

All automated RoI detection approaches proposed here handle the detection of the encrypted image region in one of five ways, which can be applied to three different input data levels. Hence, a total of 15 different combinations of automated encryption detection approaches have been evaluated based on their real world feasibility. First we would like to summarize the three input levels which have been used to evaluate the different RoI detection approaches. The first input level involves the transformation of the JPEG2000 image to the PGM file-format and its partitioning into smaller non-overlapping blocks of equal size, which are used to automatically detect the encrypted image region. The second and least promising input level handles the extraction of JPEG2000 packet-data, which is further used to identify the encrypted RoI. Finally, the third and most promising input level involves decoding the JPEG2000 image until the inverse Wavelet transformation has been applied to the JPEG2000 codestream and the partitioning of the image coefficients into smaller blocks. These are the three input levels which have been used to evaluate the five automated encryption methods. The proposed automated encryption detection approaches are Entropy, Variance, Thresholding, Sobel-Edge-Detector and Canny-Edge-Detector. Furthermore, it should be noted that all proposed automated encryption detection approaches are JPEG2000 format-compliant, do not degrade image quality and preserve file-size, as they do not change or embed any data. Although the human eye is capable of detecting encrypted picture regions easily, the automated encryption detection methods proposed here are not. All the attempts to automatically detect the encrypted image region failed due to the problem of accurately distinguishing between encrypted-

and unencrypted-image regions containing high contrast (i.e. image regions with a high gradient magnitude value). Despite the promising attempts of detecting the encrypted image regions by threshold or by the edge detector they are not accurate enough to detect all encrypted RoIs correctly. Furthermore, the automated RoI detection methods proposed here are limited to solely detecting one RoI per image, which might be not tolerated by some applications. Nevertheless, even if it would be possible to detect the encrypted image region with an accuracy of 100 %, it would not be possible to detect the encrypted codestream parts, as the encrypted RoI size might differ greatly from the correctly detected encryption-size. This is based on the fact that we used Stubhann's RoI encryption approach, which affects the whole code-block when the RoI has been encrypted. Figure 7.2a visualizes the RoI size, which is indicated by the red border and the actual encrypted image region. Therefore, determining the encrypted codestream parts by solely detecting the encrypted image region is not sufficient.

Despite the high accuracy achieved by extracting the JPEG2000 inverse Wavelet coefficients and applying a simple Threshold to detect the encrypted image blocks accurately, it is clear that perfect detection is not possible, albeit necessary for correct decryption in most cases. Therefore, there is an imminent need to store the encrypted RoI's coordinates inside the JPEG20000 file in order to have them available during decryption. As this work does not utilize the JPEG2000 feature to signal encrypted regions in the JPEG2000 codestream, other methods have been proposed to signal the encrypted image parts. All embedding approaches proposed by this work handle the embedding of the RoI encryption specification in one of four ways. The first method involves using JPEG2000 comment segments to embed the encryption specification. However, as experimental results have shown, this method has a few drawbacks. On the one hand, it changes the JPEG2000 file size, which might not be tolerated by all applications. On the other hand, embedding the encryption specification into the JPEG2000 comment segment causes the EBCOT (Embedded Block Coding with Optimal Truncation) to handle the insertion of truncation points and the optimization of the JPEG2000 packet data differently, as the length of the main header has changed. This in turn leads to the fact that decoding the image containing the embedded encryption specification leads to image quality degradation, which might not be tolerable. Hence,in order to not degrade the image quality, the embedded encryption specification must be removed prior to decoding the JPEG2000 file. Furthermore, this embedding approach has the limitation of losing the meta data (everything embedded to the com segment) when transforming the JPEG2000 image to another format (e.g.,, from JP2 to JPEG or to PGM). Similarly, cropping or any other

form of image manipulation destroys the meta data. Therefore, embedding the encryption specification into the com segment does not represent the perfect way of embedding data into the JPEG2000 codestream.
The second and third embedding approach rely on embedding the encryption specification either at the very beginning of the JPEG2000 codestream, i.e., before the first marker (before the JPEG2000 SOC marker), or at its end, i.e., after the JPEG2000 EOC marker. Adding the encryption specification in this way is not JPEG2000 format-compliant, as the standard only allows for beginning with 0xFF4F or ending with 0xFFD9. However, these embedding approaches provide the benefit of storing the encryption specification without causing any image degradation into the JPEG2000 codestream. Furthermore, embedding the encryption specification into the JPEG2000 codestream changes the JPEG2000 file size, which might not be tolerated by all applications. However, as experimental results have shown by embedding the encryption specification at the end of the JPEG2000 codestream the image viewer IrfanView Version 4.32 and the open-source JPEG2000 implementation JJ2000 Version 5.1 are able to decode the image properly, despite non JPEG2000 format-compliant. Therefore, as experimental results have shown, embedding the encryption specification at the end of the codestream is the better choice when choosing between the two non-format-compliant embedding approaches.
Finally, the fourth embedding method embeds the data into the JPEG2000 codestream by replacing some JPEG2000 image-coefficients. As this work utilizes the JPEG2000 quality progression bitstream ordering, which aligns all the image coefficients belonging to an RoI to the beginning of the codestream, the encryption specification is embedded at the end of the JPEG2000 codestream, as this part of the codestream contains solely background data. Embedding the encryption specification into the JPEG2000 codestream by replacing image coefficients offers, on the one hand the, benefit of preserving the length of the JPEG2000 codestream and, on the other hand, the benefit of being JPEG2000 format-compliant. However, the biggest drawback of this approach is the image quality degradation, which is linked to replacing image coefficients. However, as experimental results have shown, replacing only a few bytes lowers the image quality to a certain degree, which is not detectable by the human eye.

Using a number of data sets, we determined that our proposed length preserving embedding method is superior to all other proposed automated encryption detection or embedding approaches, which is due to the fact that it preserves the file-size, keeps JPEG2000 format-compliance and degrades the image quality only slightly, as experimental results have shown.

Appendix A

Source Code – Embedding Techniques

Listing A.1: Embed encryption specification into the JPEG2000 COM-segment

```
1  // Detect position of COM marker in JPEG2000 bitstream
2  int startCOM = ImageFileHelper.findCOM(inputFile);
3
4  // Move the BufferedRandomAccessFile inputFile to position after
       the COM marker
5  inputFile.seek(startCOM + 2);
6  // Read length of COM segment, minus 4 bytes (Lcom + Rcom)
7  short comLen = (short) (inputFile.readShort() - 4);
8
9  // Move inputFile to position of first payload byte
10 inputFile.seek(startCOM + 6);
11
12 // Create COM payload
13 byte[] writePayload = encRoundsTobyteArray(tmp, length);
14
15 // COM marker
16 hbuf.writeShort(COM);
17
18 // Calculate length: Lcom(2) + Rcom (2) + string's length;
19 markSegLen = 2 + 2 + str.length();
20 hbuf.writeShort(markSegLen);
21
22 // Rcom
23 hbuf.writeShort(1);
24
25 // Write the COM payload (used for multiple packet encryption)
26 hbuf.write(writePayload, 0, writePayload.length);
```

Listing A.2: Detect position of COM-marker in JPEG2000 bitstream

```
1  public static int findCOM(BEBufferedRandomAccessFile _fileHandle)
2  {
3     short data;
4     for (int i = 0; i < _fileHandle.length(); i++)
5     {
6        _fileHandle.seek(i);
7        data = _fileHandle.readShort();
8        if (data == Markers.COM)
9        {
10          return i;
11       }
12    }
13    return 0;
14 }
```

Listing A.3: Create encoded encryption specification - used to embed into the JPEG2000 COM-segment

```
1  private byte[] encRoundsTobyteArray(Vector<EncPktCounter> _tmp,
      int _lenth)
2  {
3     byte[] encRounds = new byte[2 + (3 * _tmp.size())];
4     byte[] encryptionLength = shortTobyteArray((short)_lenth);
5
6     System.arraycopy(encryptionLength, 0, encRounds, 0, 2);
7
8     // Loop over Vector of packet-ID and encryption counter
9     for (int i = 0; i < _tmp.size(); i++)
10    {
11       // Copy byte[] from  _tmp Vector to byte[] encRound
12       // with an offset of 2 bytes for the length of the
            encrypted bitstream and
13       // 3 bytes for every iteration (entry of the _tmp Vector)
14       System.arraycopy(_tmp.get(i).getbyteArray(), 0, encRounds,
            2 + i * 3, 3);
15    }
16
17    return encRounds;
18 }
```

Listing A.4: Load embedded payload from COM-segment - used to decrypt the JPEG2000 packets

```
1  private Vector<EncPktCounter> loadEncryptionRounds(int _length,
       int _start) throws Exception
2  {
3     Vector<EncPktCounter> indexRoundVector = new Vector<
          EncPktCounter>();
4     m_inputFile.seek(_start);
5     for (int i = 0; i < ((_length) / 3); i++)
6     {
7        indexRoundVector.add(new EncPktCounter(m_inputFile.
             readShort(), m_inputFile.readbyte()));
8     }
9     return indexRoundVector;
10 }
```

Listing A.5: Embed encryption specification prior to the SOC-marker

```
1  // contains the RoI encryption specification, with length of
       encrypted bitstream and all the packets with corresponding
       packet-ID that have been encrypted more than once
2  byte[] before = encRoundsTobyteArray(_tmp, _length);
3
4  // all the data of the encoded and encrypted JPEG2000 codestream
5  byte[] inputBuffer = new byte[inputFile.length()];
6  inputFile.seek(0);
7  inputFile.readFully(inputBuffer, 0, inputFile.length());
8
9  // write new file, with new header
10 outputFile.seek(0);
11 // write EMB marker, which is used to signal acitve embedding
12 outputFile.writeShort(Markers.EMB);
13 // write encryption specification
14 outputFile.write(before, 0, before.length);
15 // write unmodified JPEG2000 codestream after embedded data
16 outputFile.write(inputBuffer, 0, inputBuffer.length);
```

Listing A.6: Check whether JPEG2000 codestream starts with EMB-marker (indicates active embedding)

```
1  public static boolean embAtFront(BEBufferedRandomAccessFile
       fileHandle)
2  {
3     fileHandle.seek(0);
4     return (fileHandle.readShort() == Markers.EMB) ? true : false;
5  }
```

Listing A.7: Embed encryption specification into JPEG2000 codestream by replacing image coefficients

```
// _tmp contains the multiple encrypted packets
// _length signals the length of the encrypted bitstream
byte[] into = encRoundsTobyteArray(_tmp, _length);

// start of replacing JPEG2000 image confidents:
// total length of encoded image file - length of multiple
   encrypted packets -
// - EMB-marker (2 bytes) - SOC-marker (2 bytes)
outputFile.seek(outputFile.length() - into.length - 4);
outputFile.writeShort(Markers.EMB);
outputFile.write(into, 0, into.length);
```

Listing A.8: Extract embedded encryption specification and decrypt JPEG2000 bitstream

```
// Determine position of EMB-marker, starting to search at the
   end of the bitstream
int posEMB = ImageFileHelper.embBeforeEOC(m_inputFile);

if (posEMB > 0)
{
   // Calculate length of embedded multiple encrypted packets
   // Total length of codestream - position of EMB-marker -
   // EOC-marker (2 bytes) - length of encrypted bitstream (2
      bytes)
   int comLength = m_inputFile.length() - posEMB - 4;

   // Move m_inputFile to position after EMB-marker
   m_inputFile.seek(posEMB + 2);
   // Read length of encrypted bitstream
   int length = m_inputFile.readShort();

   // Read encryption specification - multiple encrypted packets
   // posEMB + 4 (length of encrypted bitstream and EMB-marker)
      is used as start point for reading
   Vector<EncPktCounter> tmp = loadEncryptionRounds(comLength,
      posEMB + 4);

   // decrypt encrypted JPEG2000 packets
   decryptEmbedded(length, tmp, 0);
}
```

Appendix B

Source Code – Automated RoI Detection

Listing B.1: Calculate Entropy – Specific block of data

```
1  private double calculateEntropy(int[] _values) throws Exception
2  {
3     Double entropy = 0.0, frequency = 0.0;
4     // get frequency of each entry
5     Map<Integer, Integer> map = frequency(_values);
6     // calculate the entropy
7     for (Integer sequence : map.keySet())
8     {
9        frequency = (double) map.get(sequence) / _values.length;
10       entropy -= (frequency *(Math.log(frequency)/Math.log(2)));
11    }
12    return entropy;
13 }
```

Listing B.2: Check Threshold – Specific block of data

```
1  private boolean checkThreshold(int[] _values)
2  {
3     for (int i = 0; i < _values.length; i++)
4     {
5        // check if image coefficient is within valid range
6        if (_values[i] < 0 || _values[i] > 255)
7        {
8           return true;
9        }
10    }
11    return false;
12 }
```

Listing B.3: Calculate Frequency Distribution (histogram) – Specific block of data

```
1  private Map<Integer, Integer> frequency(int[] _values)
2  {
3     Map<Integer, Integer> map = new HashMap<Integer, Integer>();
4
5     // count the occurrences of each value
6     for (int i = 0; i < _values.length; i++)
7     {
8        // in case no entry with given Integer value is present in
            HashMap, add it to Map
9        if (!map.containsKey(_values[i]))
10       {
11          map.put((int) _values[i], 0);
12       }
13       // increase counter of occurence
14       map.put((int) _values[i], map.get((int) _values[i]) + 1);
15    }
16
17    return map;
18 }
```

Listing B.4: Calculate Variance – Specific block of data

```
1  private double variance(int[] _values)
2  {
3     // get mean value of input array, used to calculate variance
4     double mean = mean(_values);
5     double variance = 0.0, frequency = 0.0;
6
7     // call method frequency, which stores the input array into a
         Map indicating the distribution of the values
8     Map<Integer, Integer> map = frequency(_values);
9
10    // calculate the variance
11    for (Integer sequence : map.keySet())
12    {
13       frequency = (double) map.get(sequence) / _values.length;
14       variance += (frequency * Math.pow((sequence - mean), 2));
15    }
16
17    return variance;
18 }
```

Listing B.5: Calculate Bounding-Box for image blocks exceeding Threshold

```
1  private static void boundigBoxInvalidEntries(Vector<Boolean>
       _invalidEntries, int _cbsHeight, int _cbsWidth, int _xBlocks,
       int _yBlocks, int _imgWidth, int _imgHeight)
2  {
3     // upper left x and y coordinates
4     int leftX = -1;
5     int upperY = -1;
6
7     // lower right x and y coordinates
8     int rightX = -1;
9     int lowerY = -1;
10
11    // loop over the input array
12    for (int ii = 0; ii < _invalidEntries.size(); ii++)
13    {
14       // check whether the current codeblock contains invalid
            numbers (indicated by true)
15       if (_invalidEntries.get(ii) == true)
16       {
17          // when entering this part for the first time, leftX is
               -1, otherwise leftX has any other value
18          if (leftX == -1)
19          {
20             // store upper left coordinates into variables
21             leftX = (ii % _xBlocks) * _cbsWidth;
22             rightX = leftX + _cbsWidth;
23
24             upperY = ((ii + 1) / _xBlocks) * _cbsHeight;
25             lowerY = upperY + _cbsHeight;
26          }
27          // bounding-box coordinates have been initialized
28          else
29          {
30             // check right corner of bounding box
31             if (rightX <= (ii % _xBlocks) * _cbsWidth)
32             {
33                rightX = ((ii%_xBlocks) * _cbsWidth) + _cbsWidth;
34             }
35             // check left corner of bounding box
36             else if (leftX > (ii % _xBlocks) * _cbsWidth)
37             {
```

```
38                leftX = (ii % _xBlocks) * _cbsWidth;
39             }
40             // check lower y coordinate of bounding box
41             if ((lowerY) <= ((ii + 1) / _xBlocks) * _cbsHeight)
42             {
43                lowerY = (((ii + 1) / _xBlocks) * _cbsHeight) +
                      _cbsHeight;
44             }
45          }
46       }
47    }
48    System.out.println("UpperLeftX:␣" + leftX + "␣UpperLeftY:␣" +
         upperY + "RightX:␣" + rightX + "LowerY:␣" + lowerY);
49 }
```

Listing B.6: Load Image Data from JPEG2000 Codestream

```
1  inputFile = ImageFileHelper.openFile(tmpFile.getPath());
2  int nextSOPpos;
3  int pos;
4  // position of SOD marker
5  pos = ImageFileHelper.findNextSOD(inputFile, 4) + 2;
6  pos = ImageFileHelper.findNextEPH(inputFile, pos);
7
8  int length = inputFile.length() - pos - 2;
9  // start packet extraction
10 while (length > 0)
11 {
12    nextSOPpos = ImageFileHelper.findNextSOP(inputFile, pos);
13
14    byte[] data = new byte[nextSOPpos - pos];
15    // read packet-data
16    inputFile.seek(pos);
17    inputFile.readFully(data, 0, data.length);
18    // shift the input data
19    int[] shiftedData = shiftDatabyteAry(data);
20
21    entropyCheck(shiftedData);
22    variance(shiftedData);
23
24    length = length - (nextSOPpos - pos);
25    pos = ImageFileHelper.findNextEPH(inputFile, (nextSOPpos+4));
26 }
```

Appendix C

Results – Embedding Techniques

This chapter depicts some figures, illustrating the experimental results, obtained by embedding the encryption specification into the JPEG2000 codestream. Therefore, the shown figures have the following configuration in common, unless stated otherwise. All figures show the average of 1000 simulations and the x-axis gives information about the parameterization used to encode the JPEG2000 image: Wavelet-level (ranging from 0 - 3) and code-block-size (containing the following sizes: 4×4, 8×8, 16×16, 32×32 and 64×64 pixel). All evaluations are based on the following surveillance camera images from the SCFace image database [46] (given in brackets are the image-, RoI-size and its upper left corner coordinates): cam1_1 (75×100 pixel, 36×72 pixel, x = 27, y = 12 pixel), cam3_3 (168×224 pixel, 87×140 pixel, , x = 27, y = 37 pixel), cam5_3 (480×640 pixel, 90×158 pixel, x = 252, y = 392 pixel) and 001_frontal (768×1024 pixel, 579×804 pixel, x = 104, y = 36 pixel).

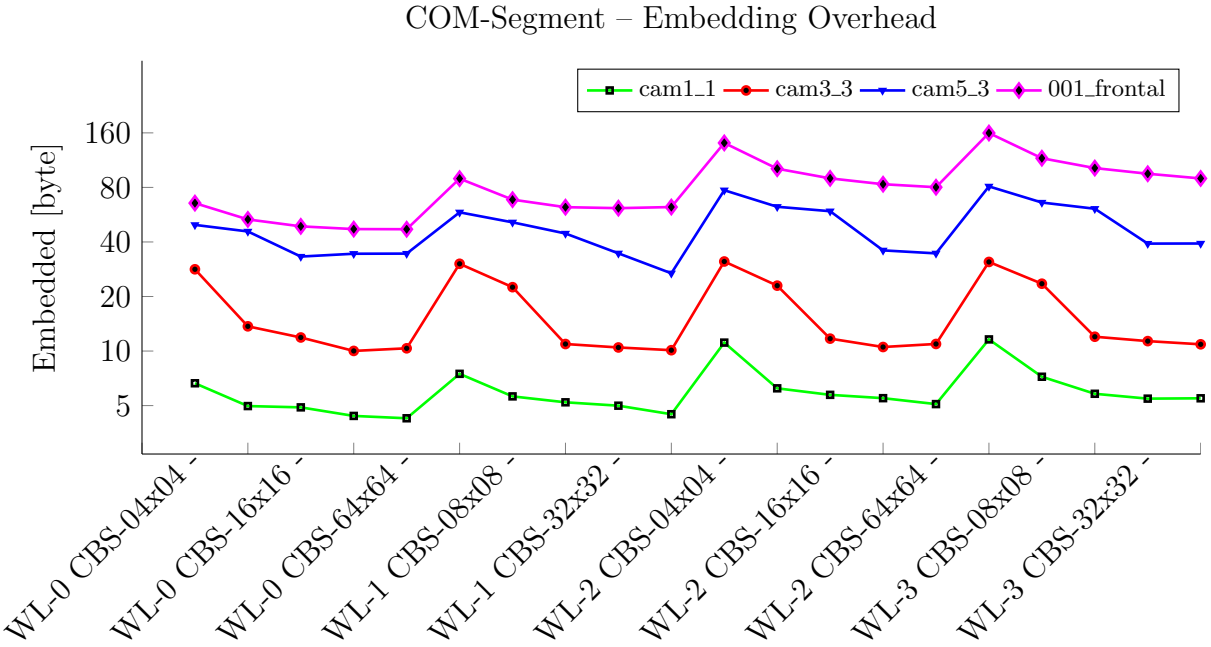


Figure C.1: This figure shows the embedding overhead caused by embedding the encryption specification into the JPEG2000 COM-segment. The y-axis shows the additional overhead, in byte, required to store the encryption specification (see Subsection 6.2.1 for further details about the used packet-structure).

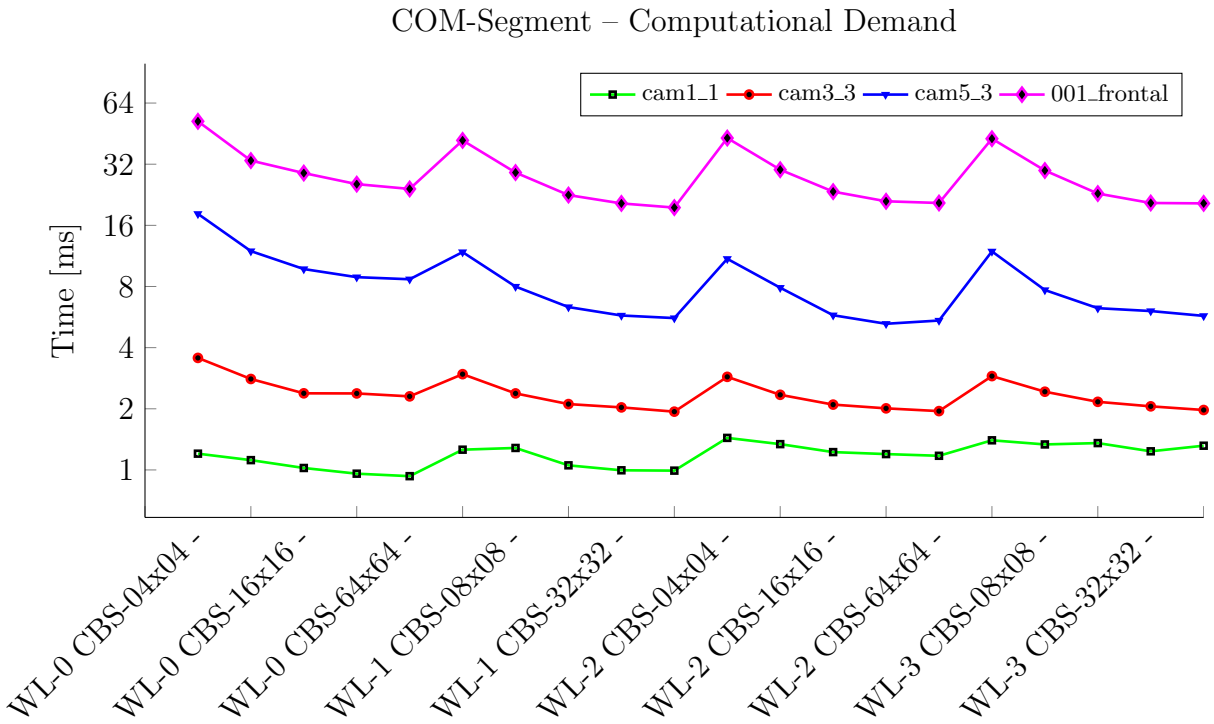


Figure C.2: This figure shows the time, in milliseconds, required to embed the encryption specification into the JPEG2000 COM-segment.

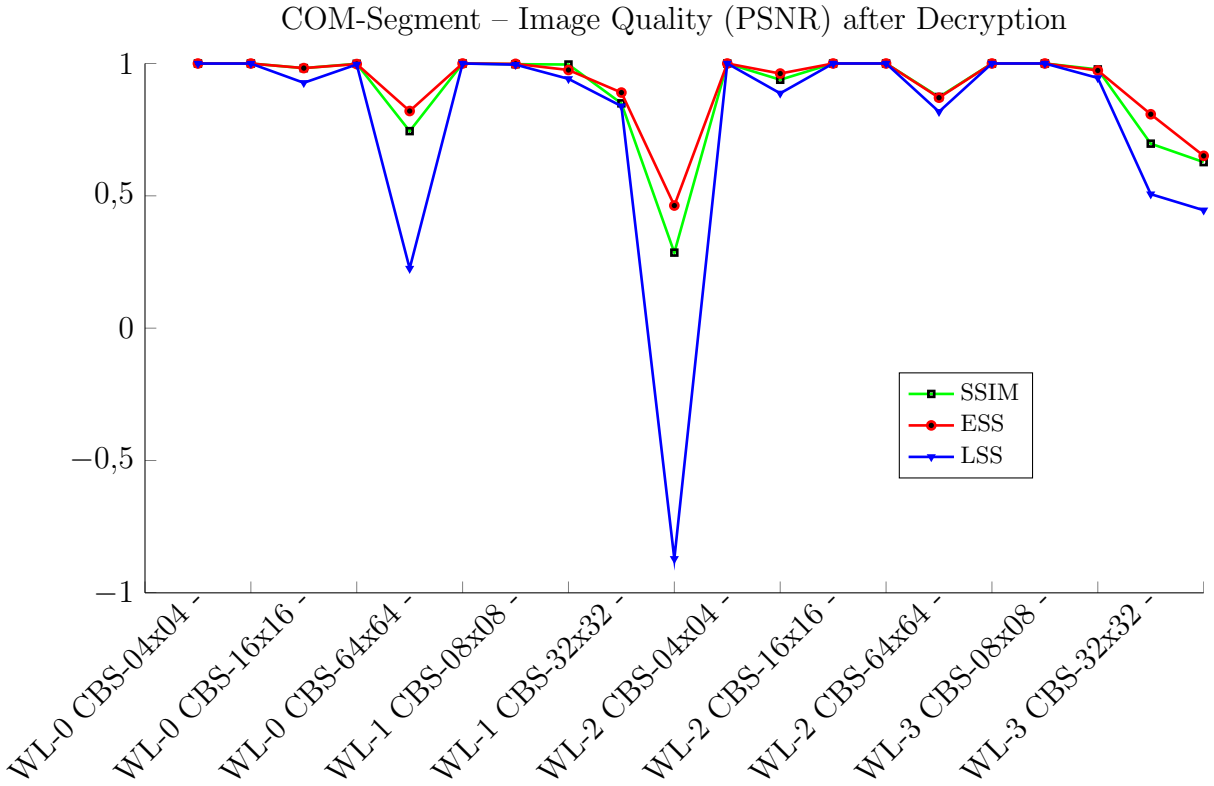


Figure C.3: This figure shows the three image quality metrics, SSIM, ESS and LSS, described in Subsection 3.2.6. All measurements are based on the COM-segment data embedding method without removing the embedded encryption specification prior to decoding the JPEG2000 image cam1_1 (see Figure 7.1a). The y-axis shows the following image quality scores: SSIM, LSS and ESS (value 1 indicates that the images are identical).

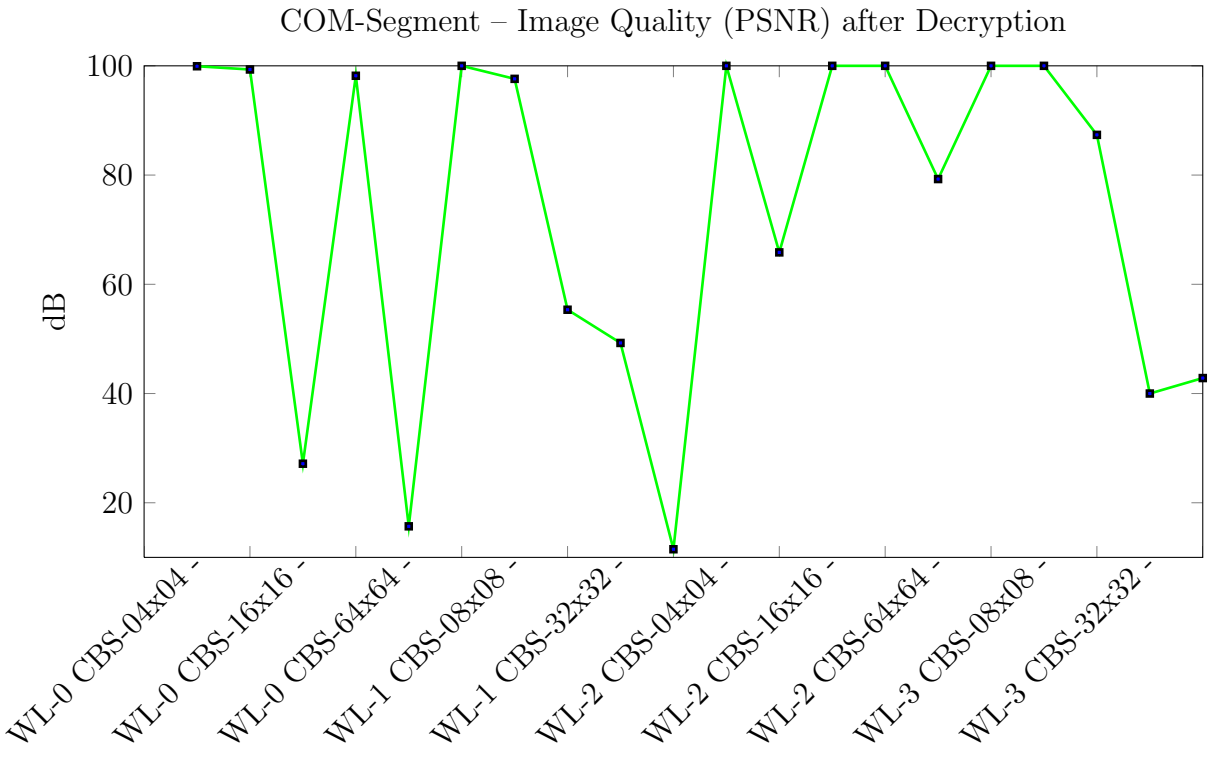


Figure C.4: This figure shows the image quality metric PSNR, in dB, after decrypting the JPEG2000 image cam1_1 (see Figure 7.1a). All the measurements are based on embedding encryption specification into the JPEG2000 COM-segment. The y-axis shows the image quality in dB, whereby a higher value indicates a higher image similarity (100 % identical images result in a PSNR of ∞, which is depicted by the value 100).

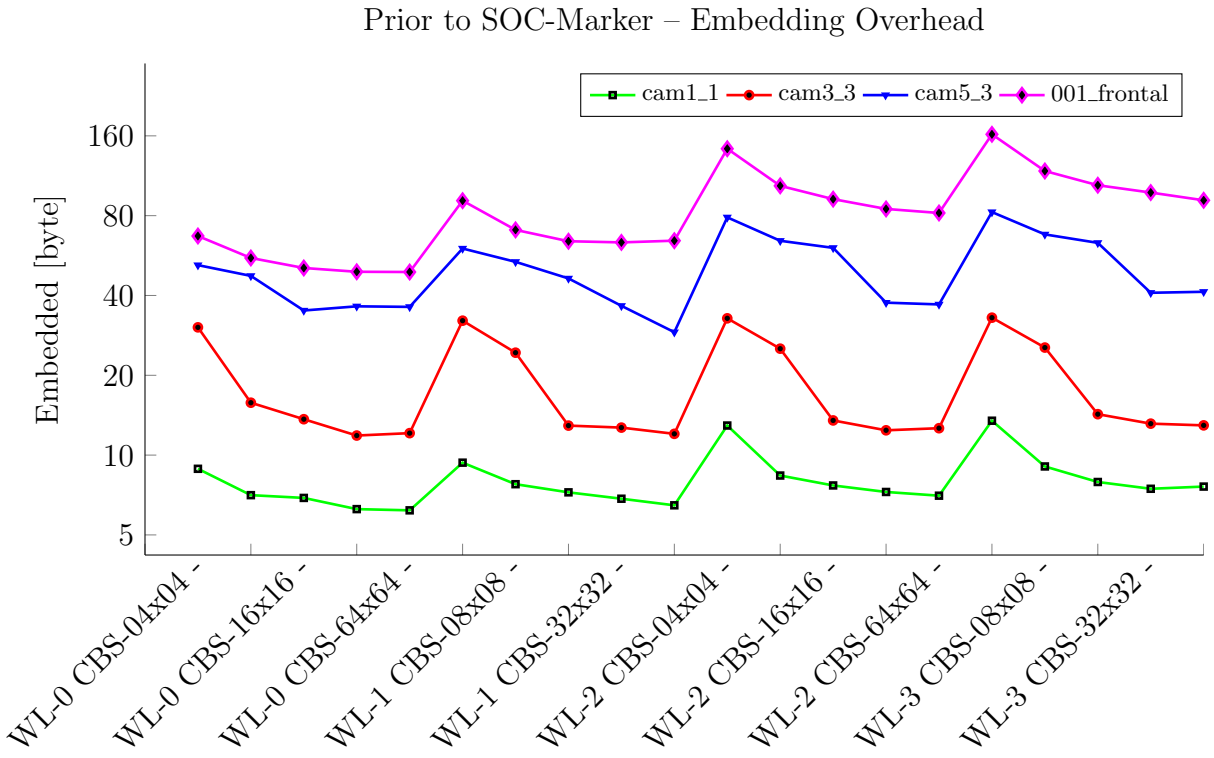


Figure C.5: This figure shows the embedding overhead caused by embedding the encryption specification prior to the JPEG2000 SOC-marker. The y-axis shows the additional overhead, in byte, required to store the encryption specification (see Subsection 6.2.2 for further details about the used packet-structure).

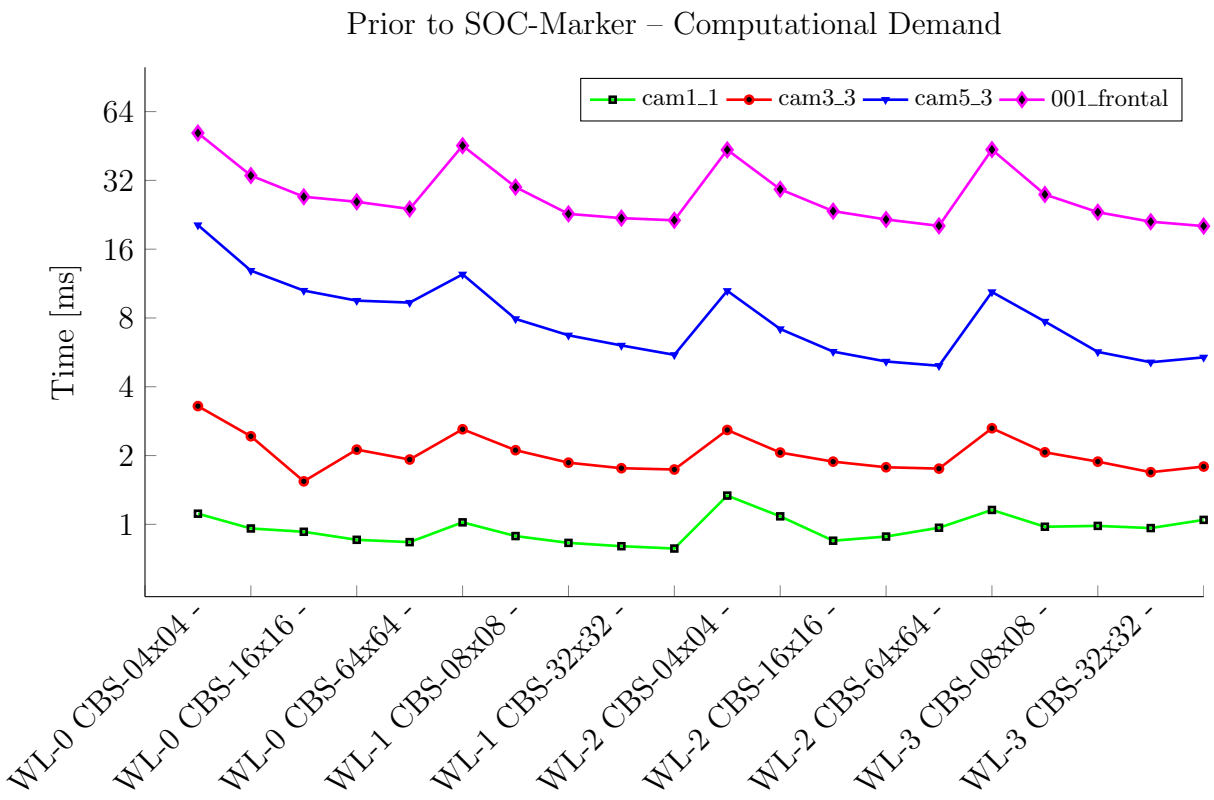


Figure C.6: This figure shows the time, in milliseconds, required to embed the encryption specification prior to the JPEG2000 SOC-marker.

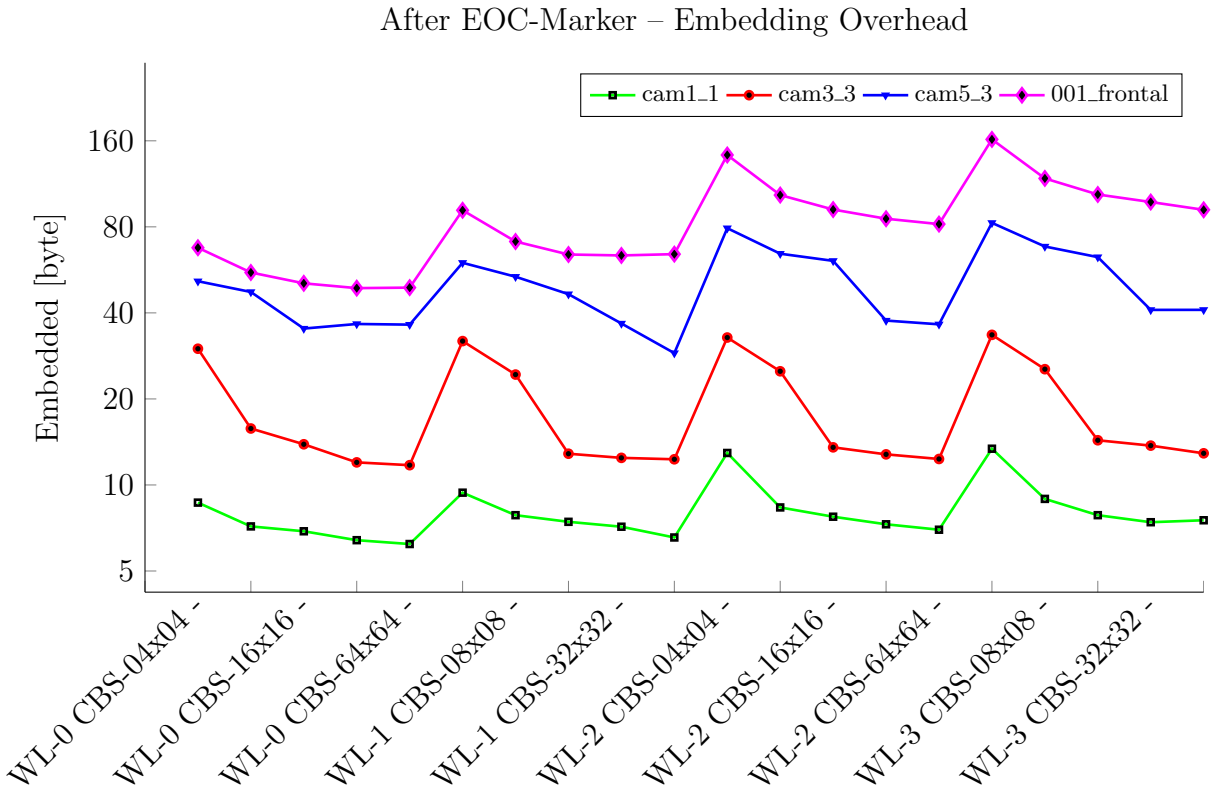


Figure C.7: This figure shows the embedding overhead caused by embedding the encryption specification after the JPEG2000 EOC-marker. The y-axis shows the additional overhead, in byte, required to store the encryption specification (see Subsection 6.2.3 for further details about the used packet-structure).

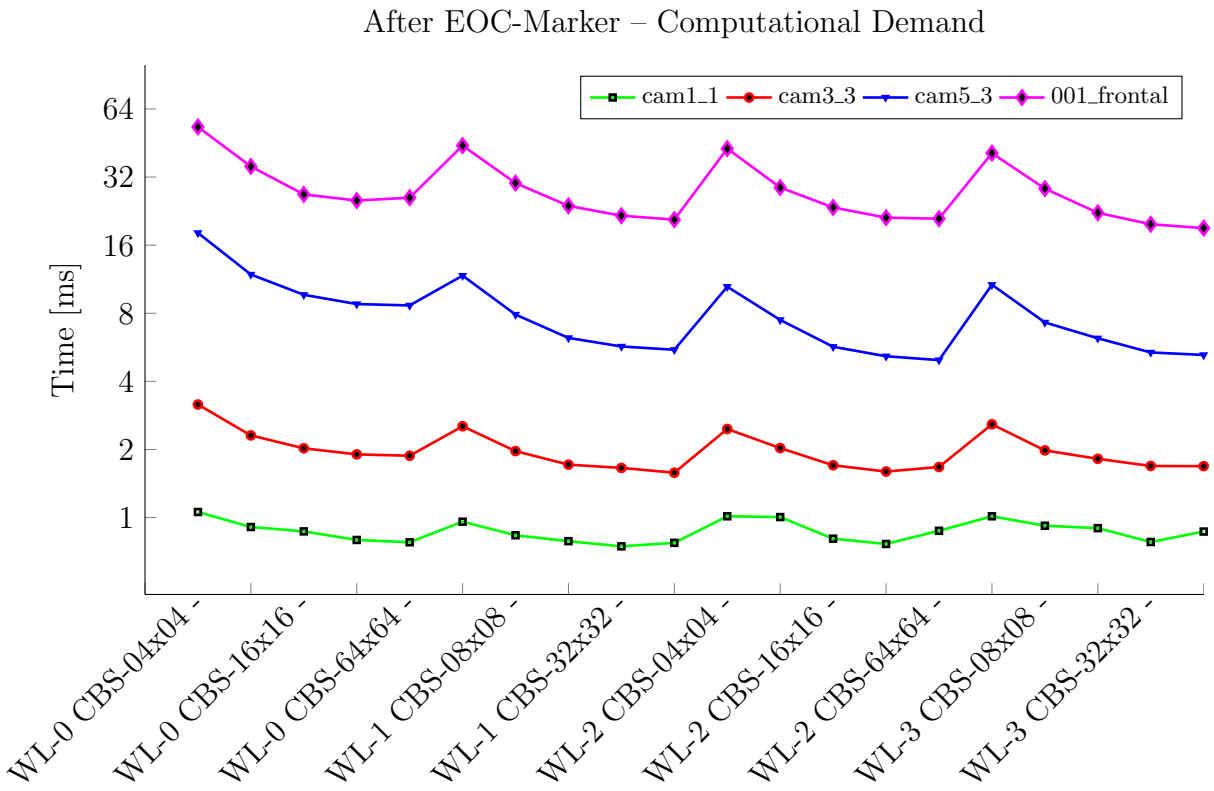


Figure C.8: This figure shows the time, in milliseconds, required to embed the encryption specification after the JPEG2000 EOC-marker.

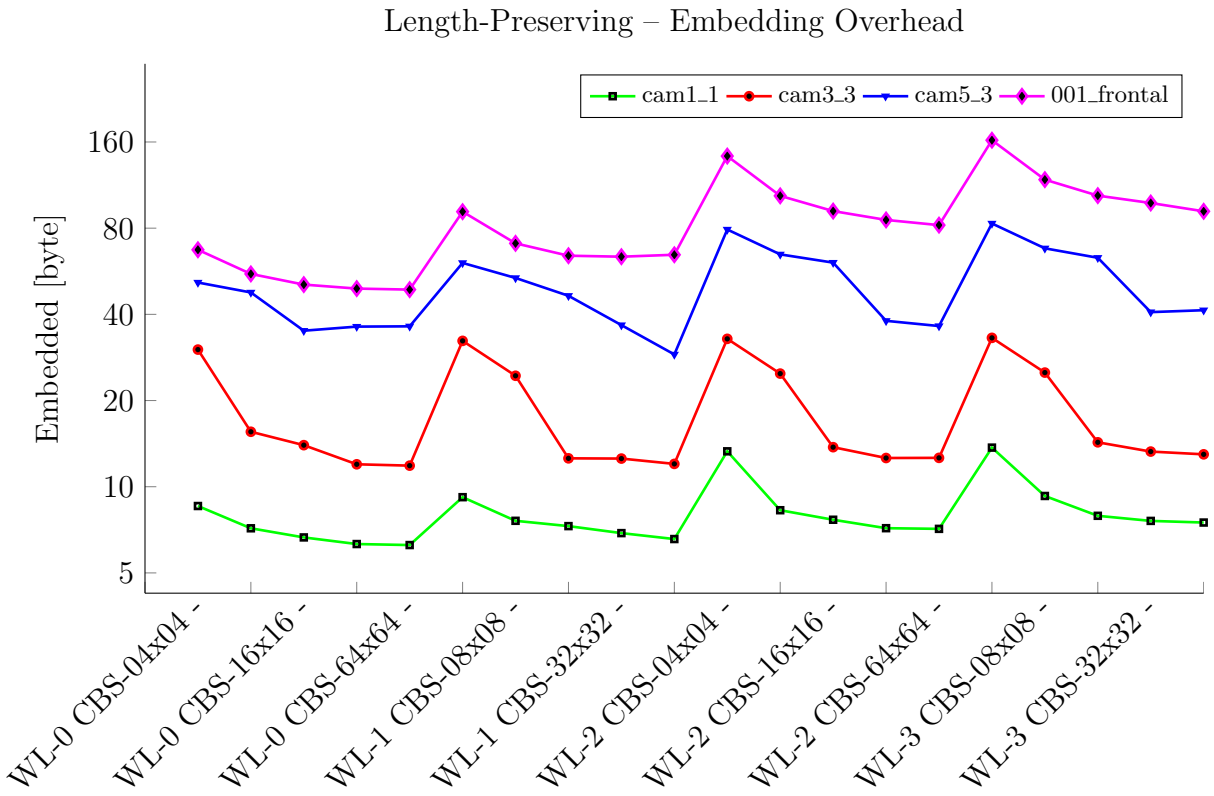


Figure C.9: This figure shows the embedding overhead caused by embedding the encryption specification into JPEG2000 codestream, by replacing image coefficients. The y-axis shows the additional overhead, in byte, required to store the encryption specification (see Subsection 6.2.4 for further details about the used packet-structure).

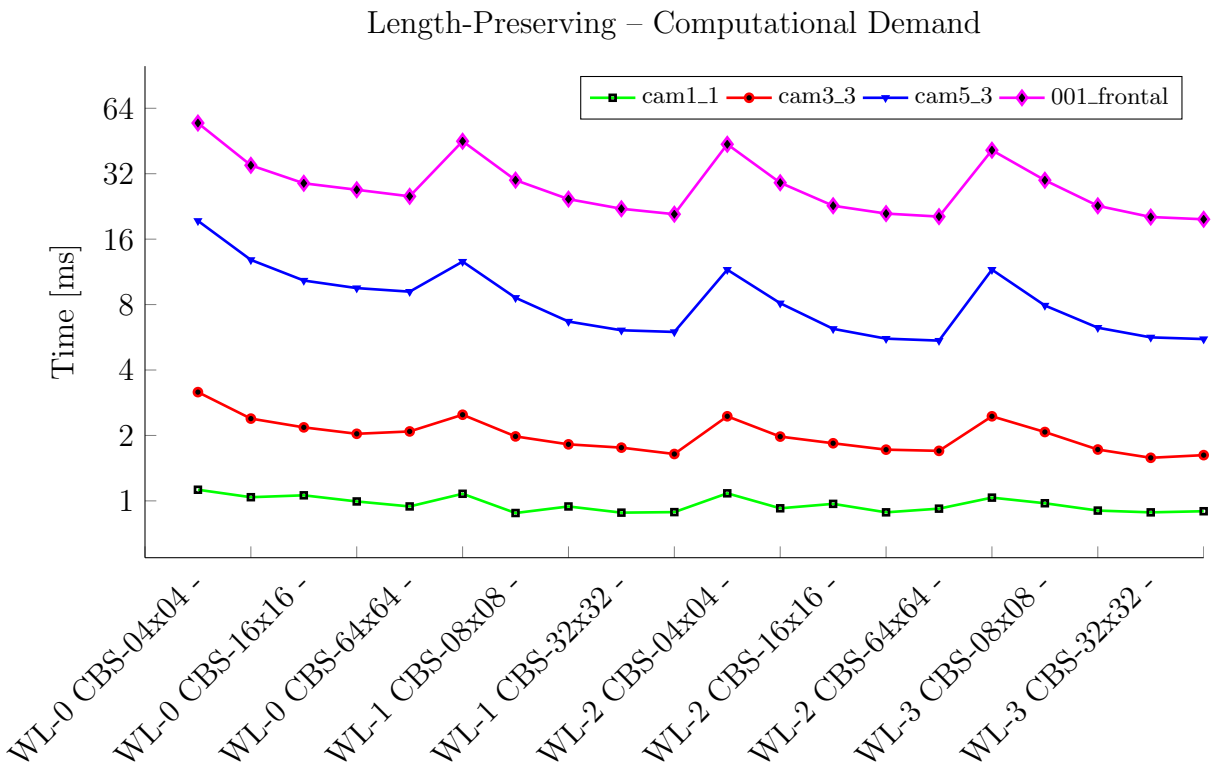


Figure C.10: This figure shows the time, in milliseconds, required to embed the encryption specification into the JPEG2000 codestream by replacing image coefficients

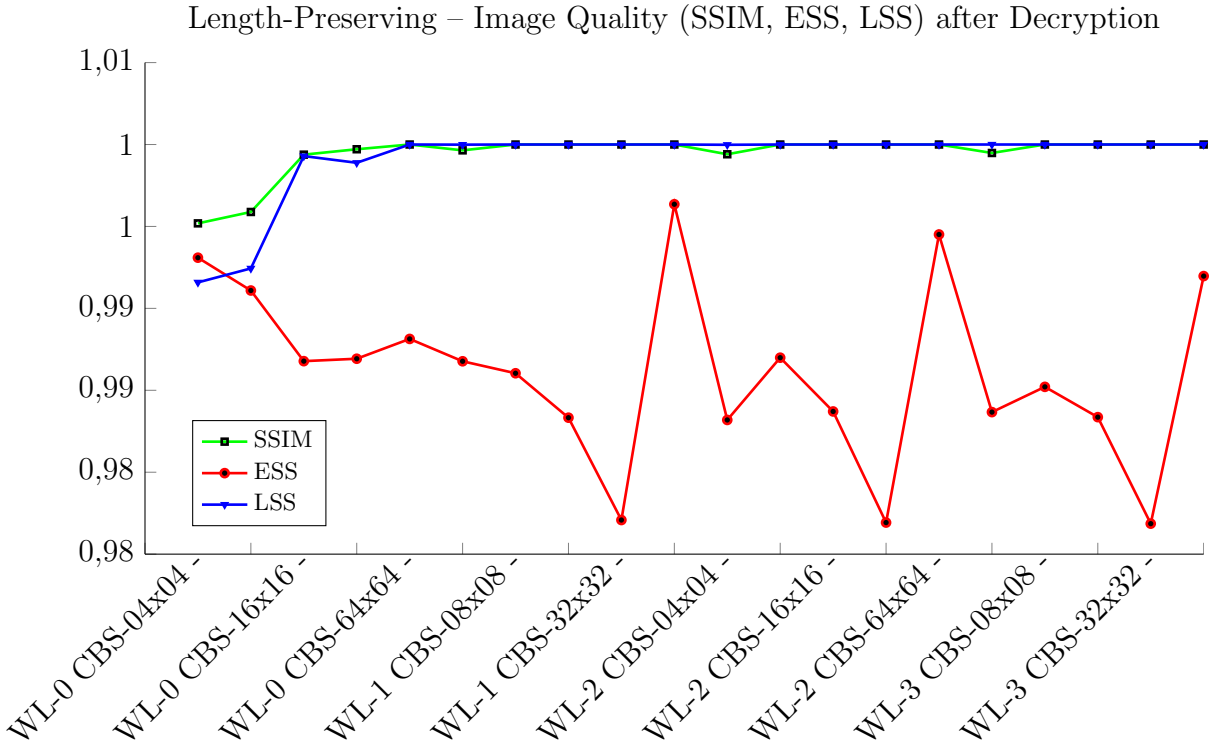


Figure C.11: This figure shows the three image quality metrics, SSIM, ESS and LSS, described in Subsection 3.2.6. All measurements are based on embedding encryption specification into the JPEG2000 codestream, by replacing the JPEG2000 image coefficients. Therefore, the image-degradation effect of embedding the encryption specification is shown. The y-axis shows the following image quality scores: SSIM, LSS and ESS (value 1 indicates that the images are identical). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

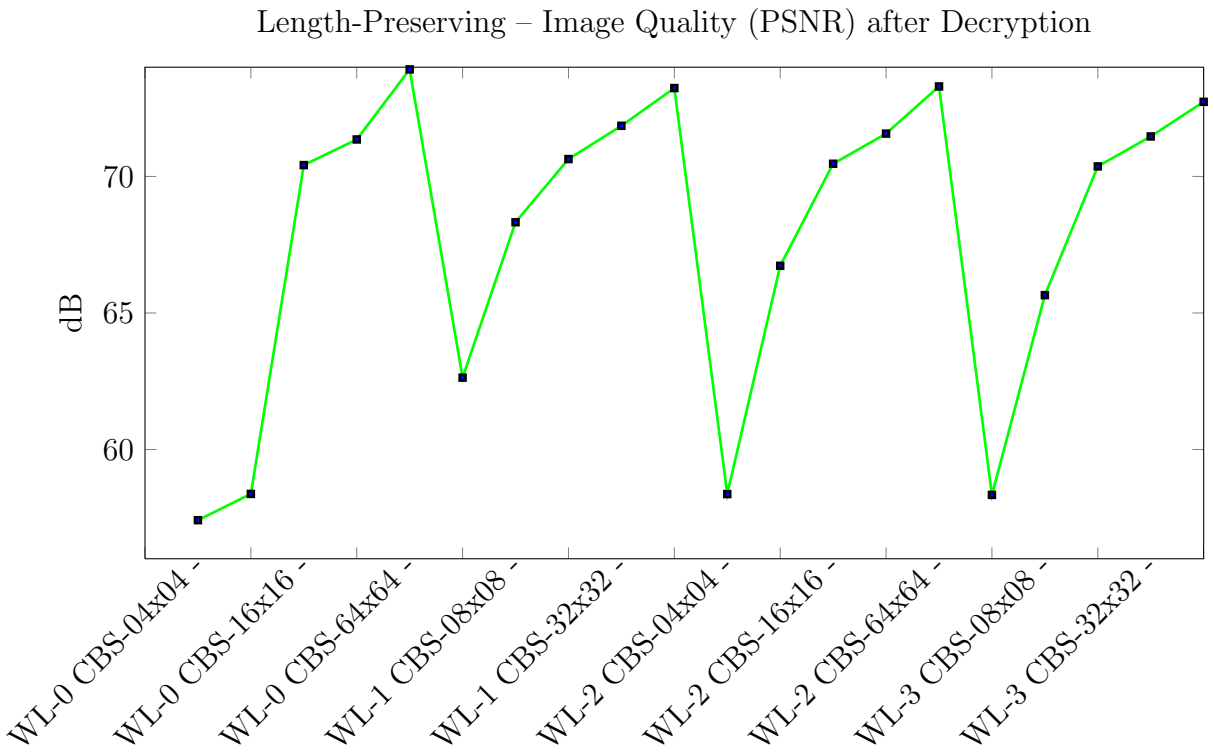


Figure C.12: This figure shows the image quality metric PSNR, in dB, after decrypting the JPEG2000 image. All the measurements are based on embedding encryption specification into the JPEG2000 codestream, by replacing image coefficients. The y-axis shows the image quality in dB, whereby a higher value indicates a higher image similarity (100 % identical images result in a PSNR of ∞, which is depicted by the value 100). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

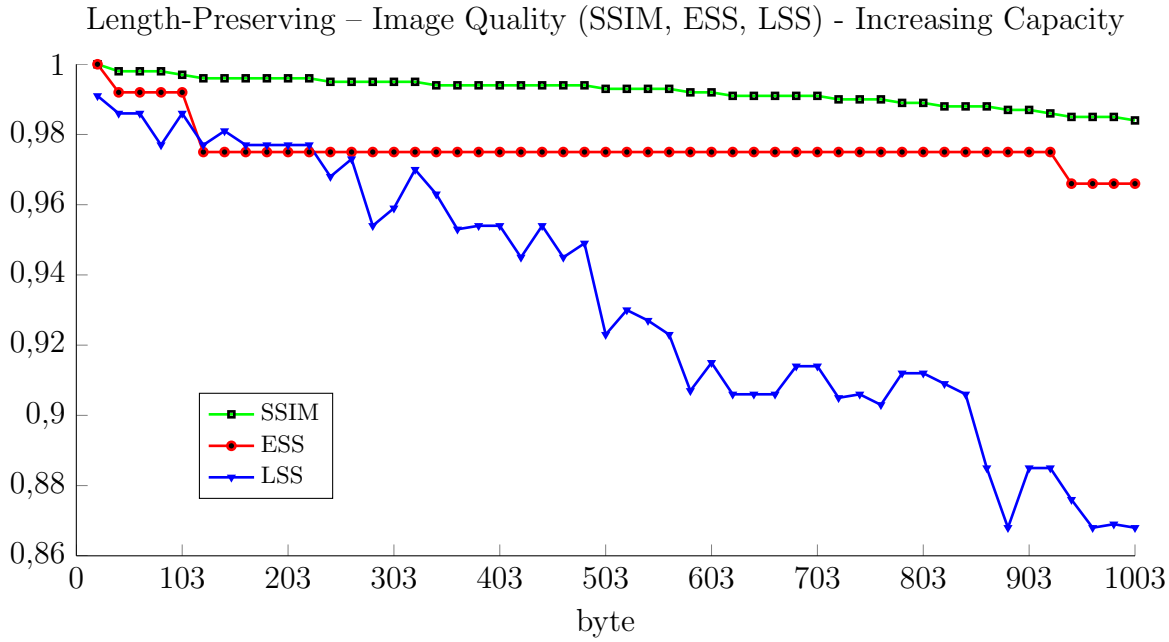


Figure C.13: This figure shows the three image quality metrics, SSIM, ESS and LSS, described in Subsection 3.2.6. All measurements are based on embedding encryption specification into the JPEG2000 codestream, by replacing the JPEG2000 image coefficients. Therefore, the image-degradation effect of embedding up to 1003 byte is shown. The y-axis shows the following image quality scores: SSIM, LSS and ESS (value 1 indicates that the images are identical). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

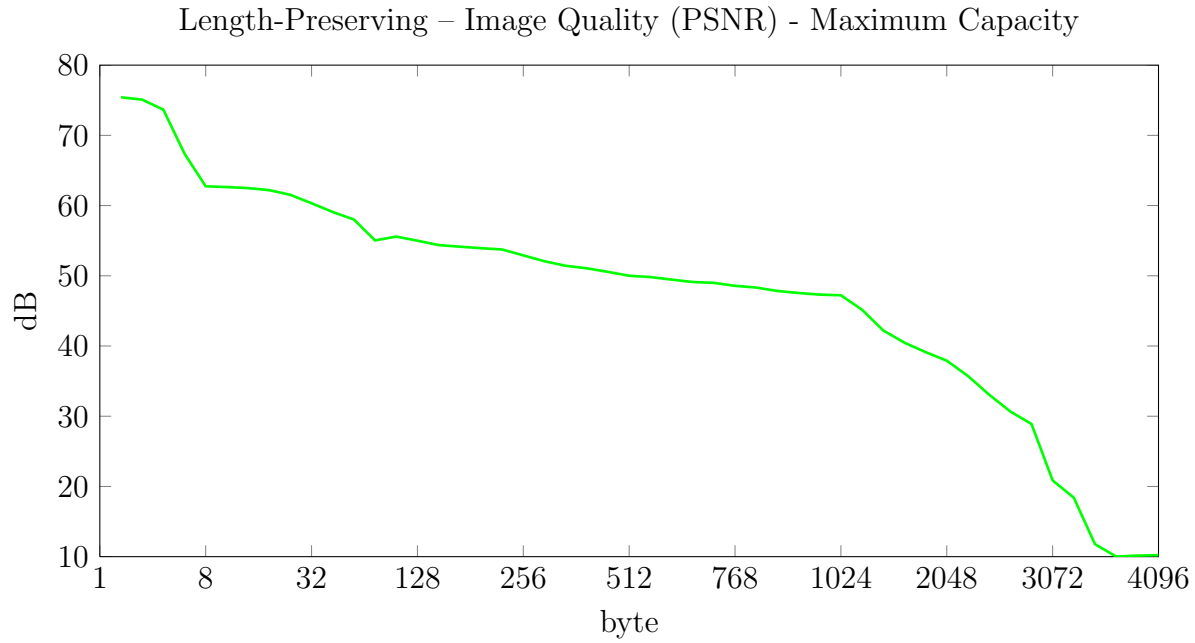


Figure C.14: This figure shows the PSNR, in dB, caused replacing up to 4096 byte of JPEG2000 image coefficients (total capacity of sample image cam1_1 is 5921 byte). The x-axis shows the data-volume, in bytes, embedded into the JPEG2000 codestream. The y-axis shows the image quality in dB, whereby a higher value indicates that the images are more similar (100% identical images result in a PSNR of ∞). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

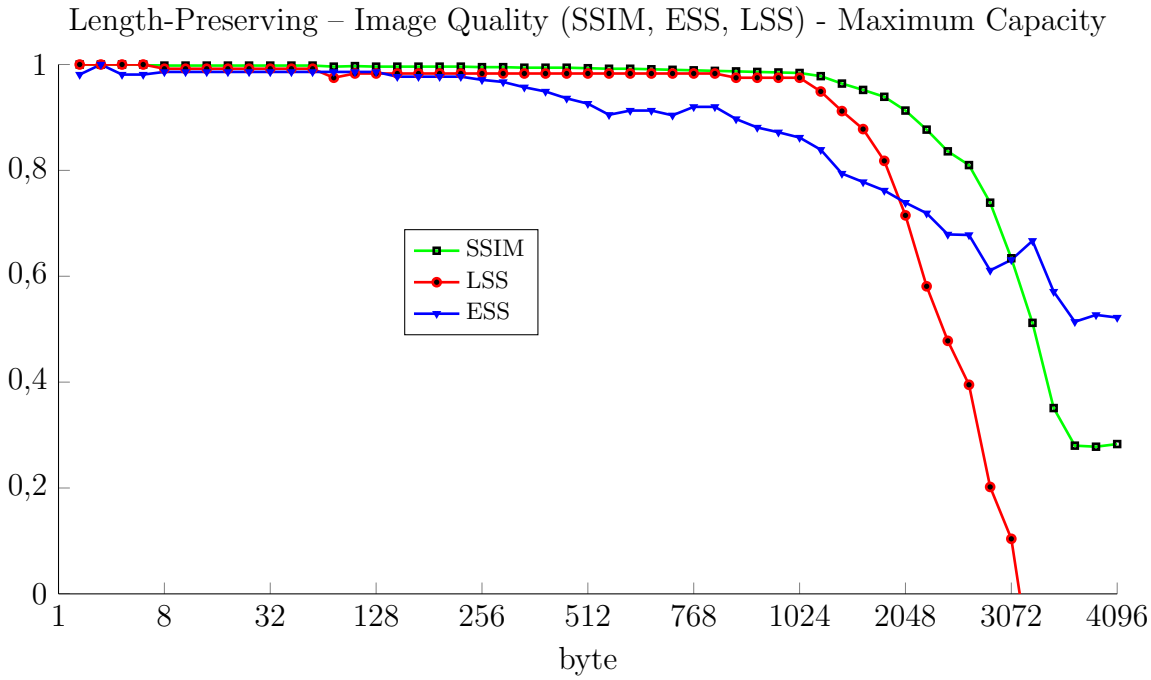


Figure C.15: This figure shows the three image quality metrics, SSIM, ESS and LSS, described in Subsection 3.2.6. All measurements are based on embedding encryption specification into the JPEG2000 codestream, by replacing the JPEG2000 image coefficients. Therefore, the image-degradation effect of embedding up to 4096 byte is shown. The y-axis shows the following image quality scores: SSIM, LSS and ESS (value 1 indicates that the images are identical). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

(a) 0 byte **(b)** 512 byte **(c)** 2048 byte **(d)** 4096 byte **(e)** 5921 byte

Figure C.16: The figures above show image cam1_1 from the SCFace image database [46], which is used to visualize the impact of replacing more and more JPEG2000 image coefficients. The image is encoded with a CBS = 16×16 pixel and a WL = 0, Image size = 75×100 pixel, the resulting JPEG2000 encoded file-size = 6718 byte and 5921 byte are used as packet body data. Figure C.16a depicts the original source image (no embedding). Figure C.16b depicts the result of embedding 512 byte into the JPEG2000 codestream, by replacing image coefficients. Figure C.16e depicts the result of replacing all the packet body data used to store the JPEG2000 encoded image coefficients.

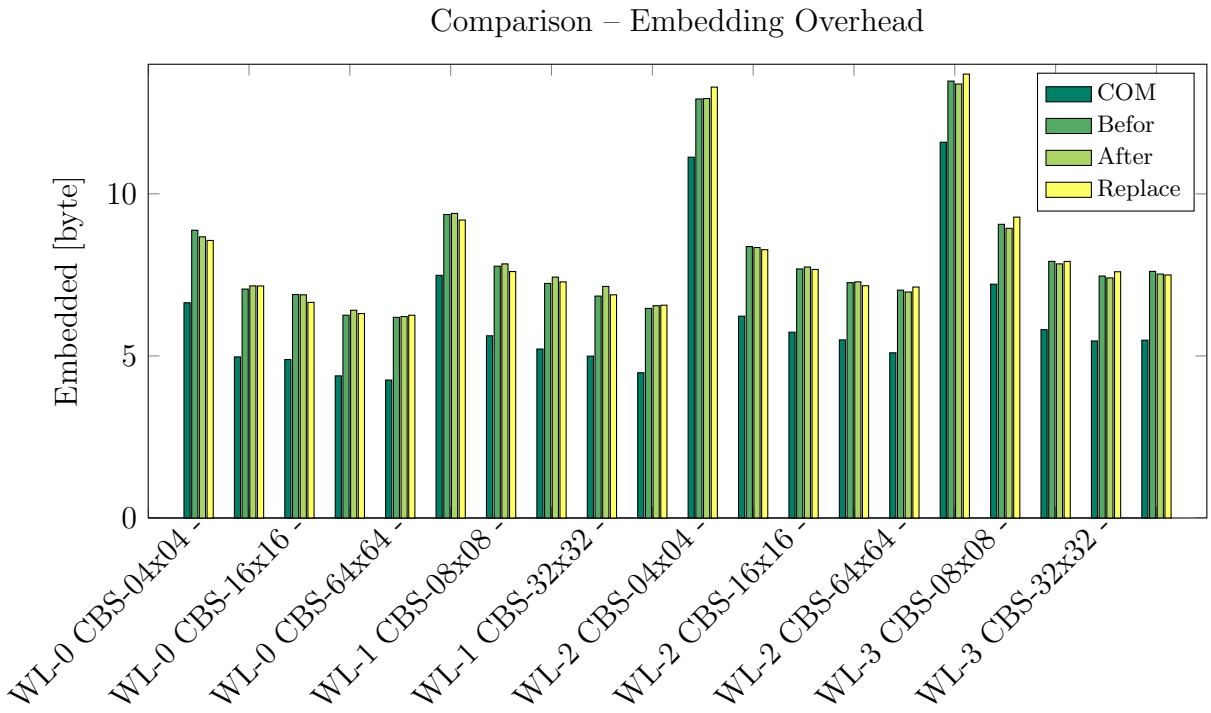


Figure C.17: This figure shows a comparison of the embedding overhead caused by the four proposed embedding methods. The y-axis depicts the embedding overhead, in byte, required to store the encryption specification into the JPEG2000 codestream (see Section 6.2 for further details about the used packet-structures). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

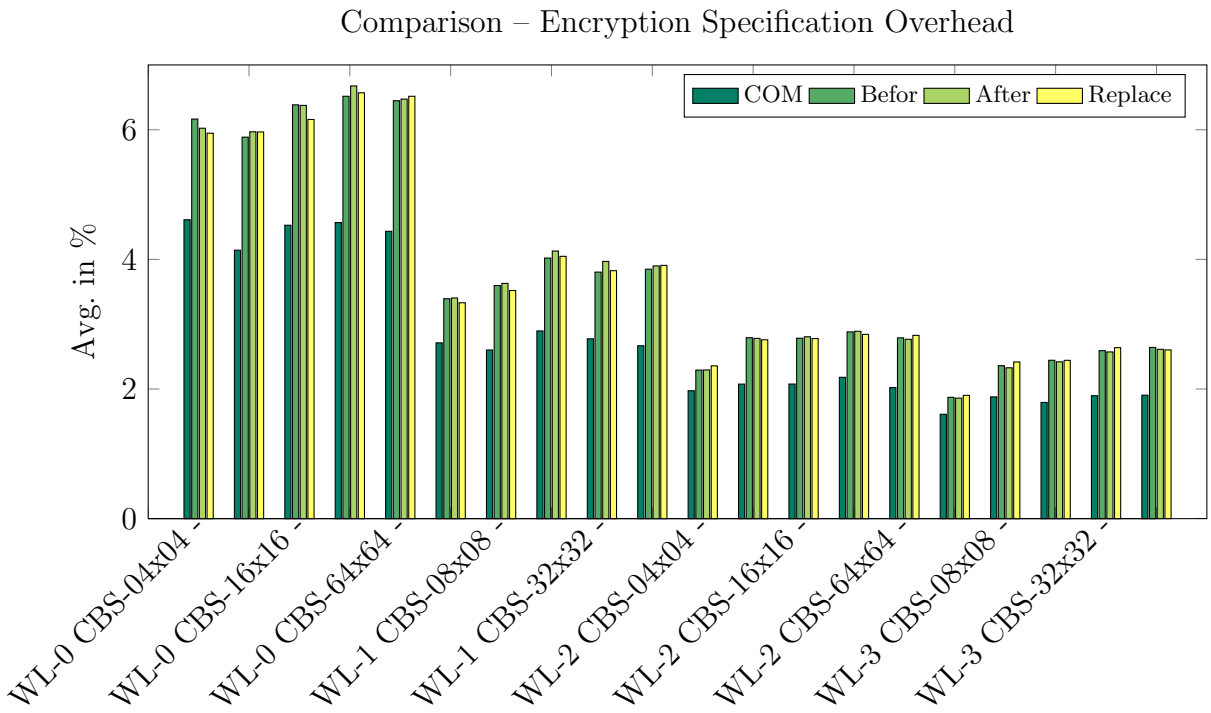


Figure C.18: This figure shows a comparison of the embedding overhead caused by the four proposed embedding methods, in relationship with storing all start-, end-values and the encryption-counters into an additional file. The y-axis shows the percentage of overhead required to store the encryption specification, compared to storing all encryption relevant data. All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

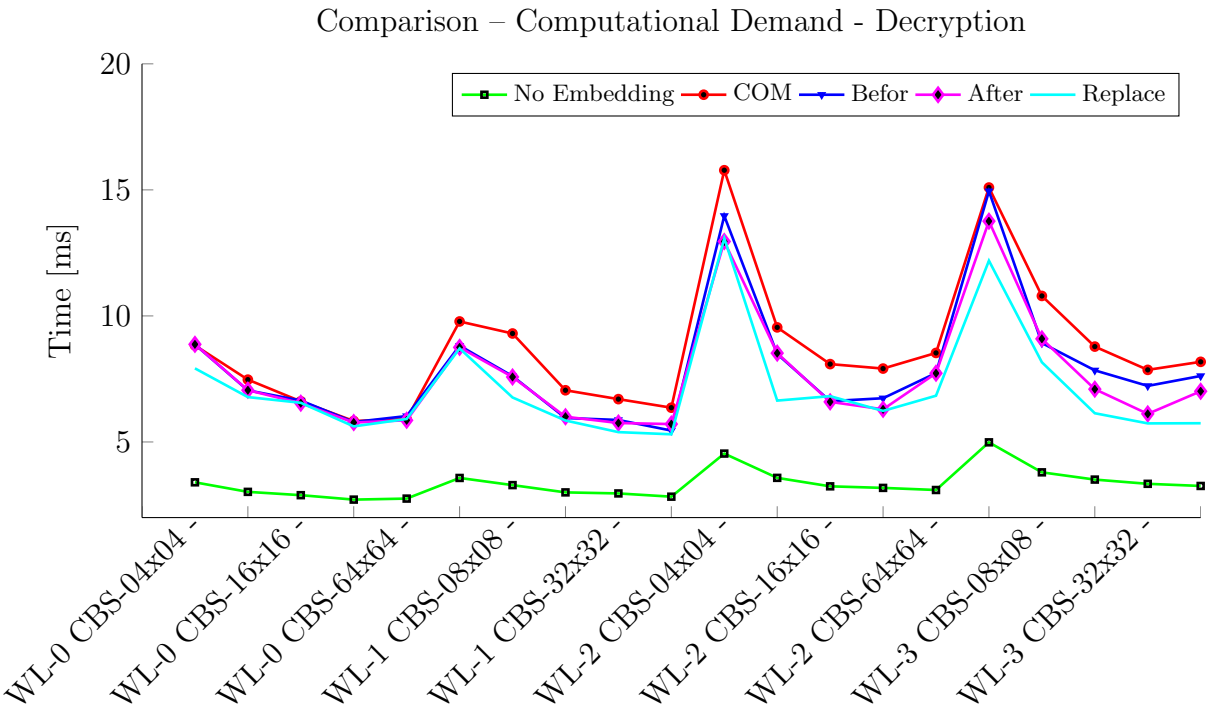


Figure C.19: The figure above compares the time, in milliseconds, required to decrypt the JPEG2000 image. Therefore, the four embedding approaches proposed by this work are compared with decrypting the JPEG2000 image, when all the start-, end-values and the encryption-counters are stored in an additional file. Hence, no additional extracting and parsing has to be performed prior to decrypting the image. All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

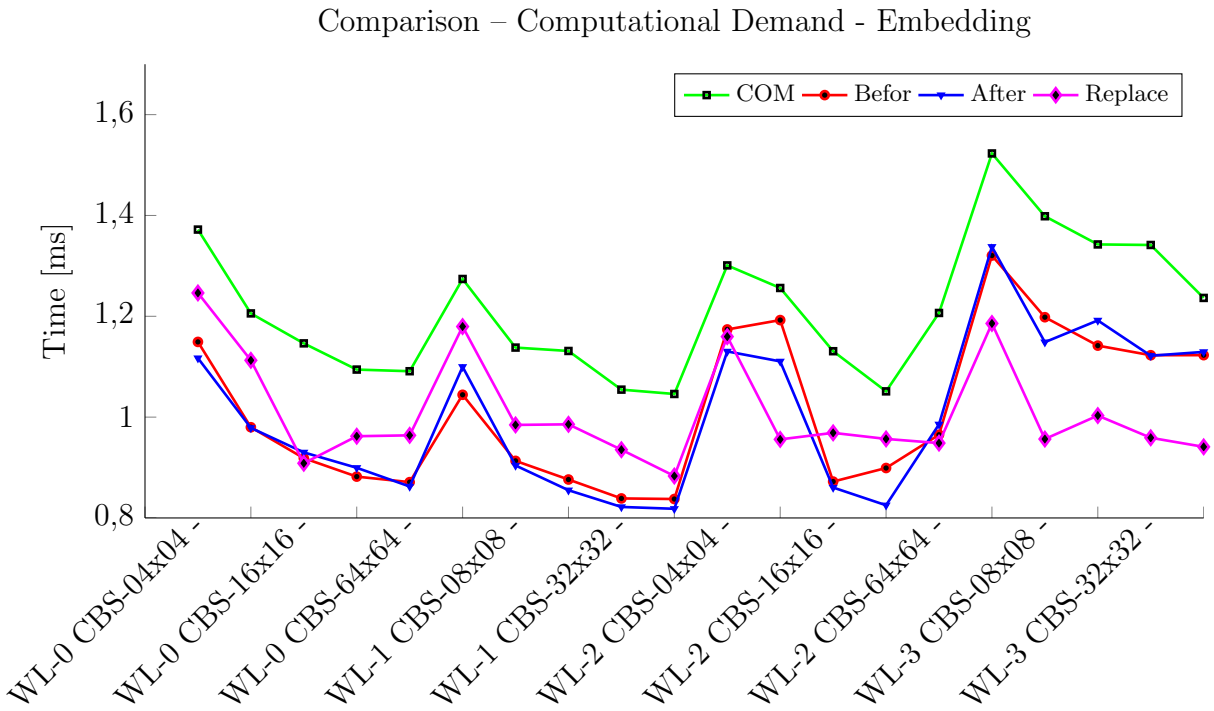


Figure C.20: The figure above compares the time, in milliseconds, required to embed the encryption specification into the JPEG2000 codestream. Therefore, the four embedding approaches proposed by this work are compared (see Chapter 6 for further details about the embedding procedure). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

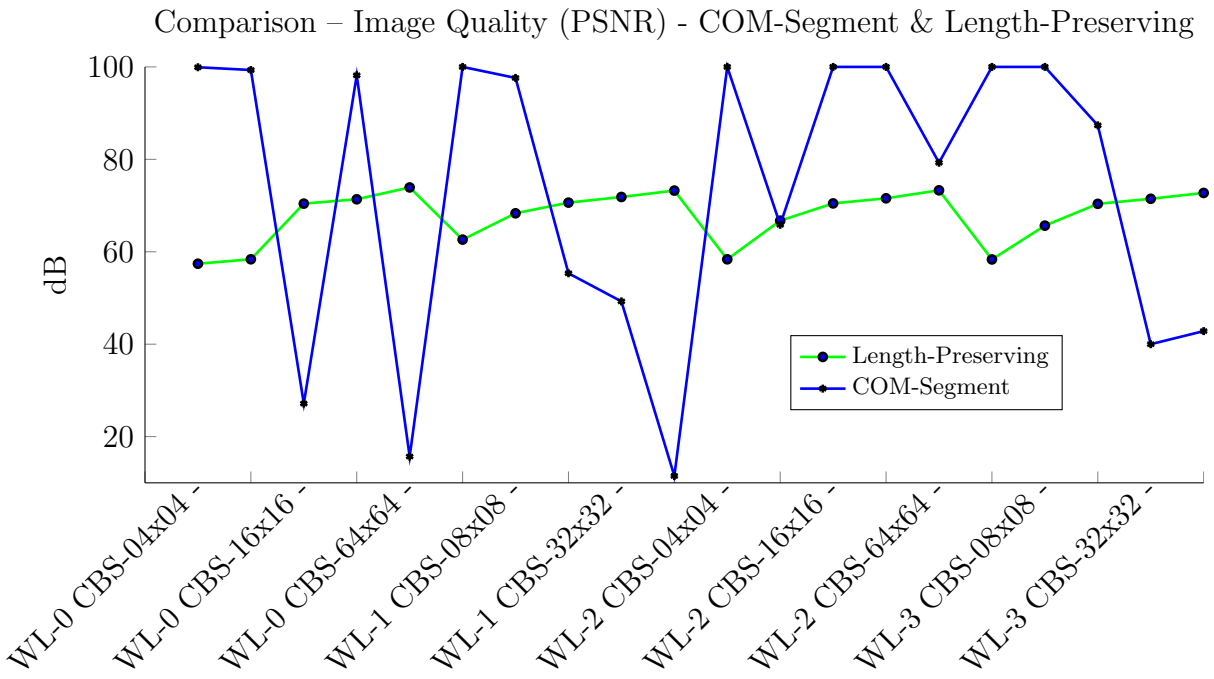


Figure C.21: This figure compares the PSNR scores, in dB, after decrypting the JPEG2000 image. Therefore, the two embedding methods proposed by this work, causing image-degradation are compared. These methods are: embedding encryption specification into the JPEG2000 COM-segment (embedded payload is not removed prior to decoding the image) or into the JPEG2000 codestream by replacing image coefficients. The y-axis shows the image quality in dB, whereby a higher value indicates a higher image similarity (100% identical images result in a PSNR of ∞, which is depicted by a dB value of 100). All measurements are based on image cam1_1 (see Figure 7.1a) from the SCFace image database [46].

Appendix D

Results – Automated RoI Detection

This chapter depicts some figures, illustrating the experimental results, obtained by automatically detecting the encrypted image regions. Therefore, the shown figures have the following configuration in common, unless stated otherwise. All the shown figures show the average of 1000 simulations and the x-axis gives either information about the used code-block-size (containing the following sizes: 4×4, 8×8, 16×16, 32×32 and 64×64 pixel) or the image block number (see Figure 7.3 for details about partitioning the image). Furthermore all images are encoded with Wavelet-level of 0, due to the reasons outlined in Section 7.4. The following surveillance camera images from the SCFace image database [46] have been used (given in brackets are the image-, RoI-size and its upper left corner coordinates): cam1_1 (75×100 pixel, 36×72 pixel, x = 27, y = 12 pixel), cam3_3 (168×224 pixel, 87×140 pixel, , x = 27, y = 37 pixel), cam5_3 (480×640 pixel, 90×158 pixel, x = 252, y = 392 pixel) and 001_frontal (768×1024 pixel, 579×804 pixel, x = 104, y = 36 pixel).

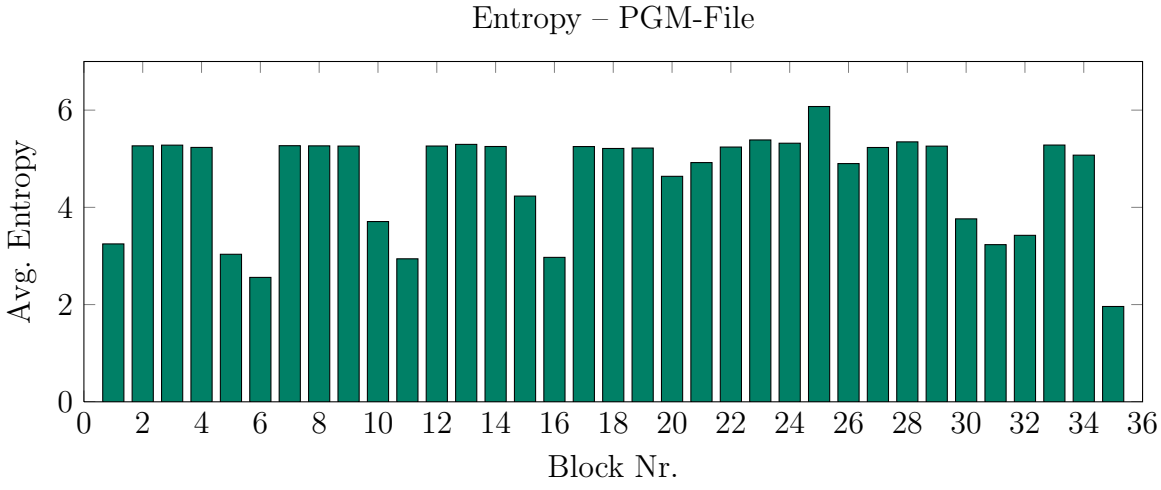


Figure D.1: The figure above shows the entropy based on image cam1_1 from the SCFace image database [46]. Therefore, the image has been converted, for testing purposes, into the PGM-file-format. Furthermore, the image is partitioned into 35 non-overlapping rectangular image blocks (see Figure 7.3). Hence, the entropy is calculated on image block basis and visualized in the figure above. The y-axis shows the average entropy for each image block. As evident from Figure 7.3 the following image blocks are unencrypted: 1, 5, 6, 10, 11, 15, 16, 20, 21, 25, 26, 30, 31, 32, 33, 34 and 35.

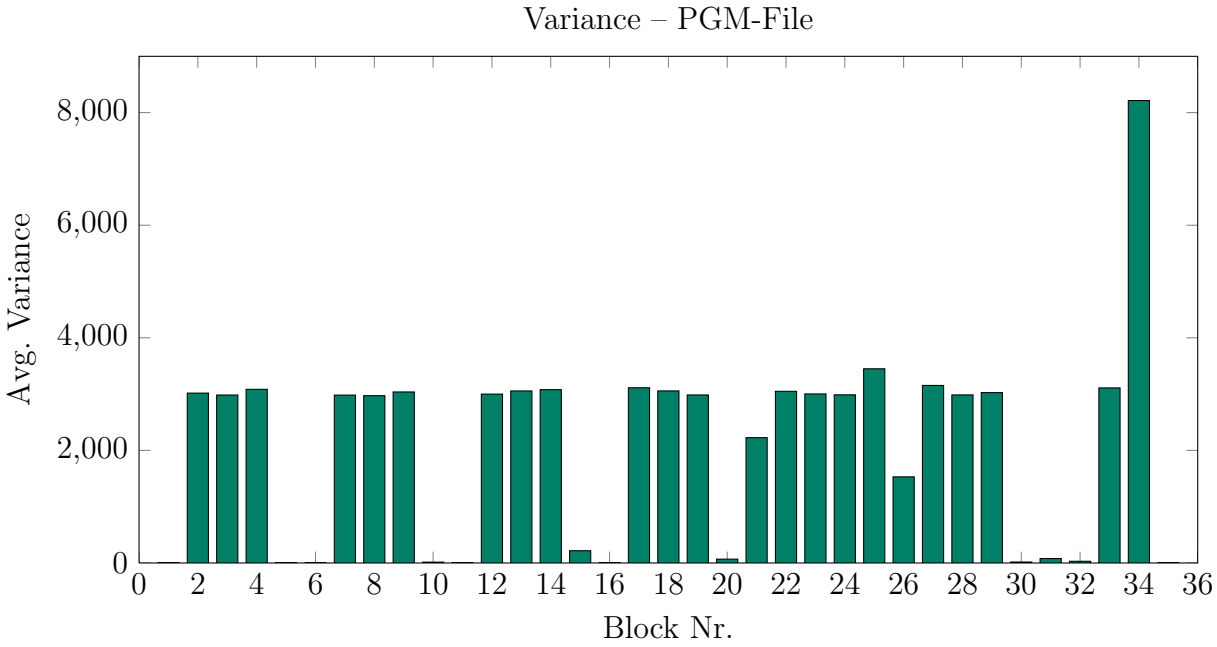


Figure D.2: The figure above shows the variance based on image cam1_1 from the SCFace image database [46]. Therefore, the image has been converted, for testing purposes, into the PGM-file-format. Furthermore, the image is partitioned into 35 non-overlapping rectangular image blocks (see Figure 7.3). Hence, the variance is calculated on image block basis and visualized in the figure above. The y-axis shows the average variance for each image block. As evident from Figure 7.3 the following image blocks are unencrypted: 1, 5, 6, 10, 11, 15, 16, 20, 21, 25, 26, 30, 31, 32, 33, 34 and 35.

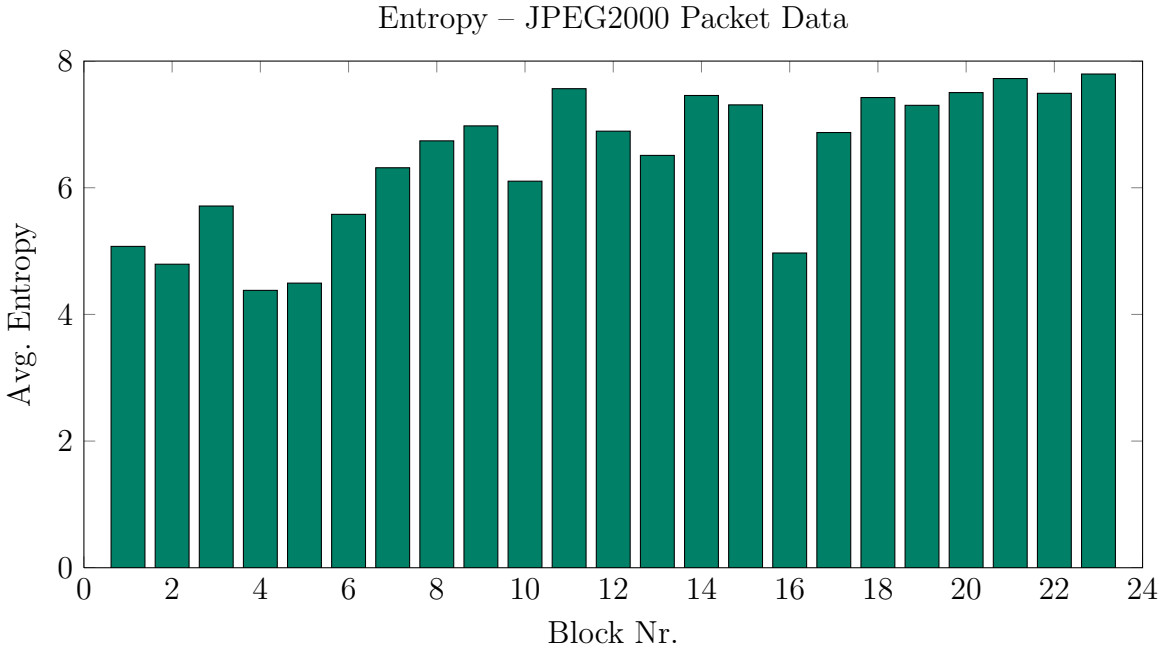


Figure D.3: The figure above shows the entropy based on image cam1_1 from the SCFace image database [46]. Therefore, the image is converted into the JPEG2000 file-format. Afterwards each JPEG2000 data-packet is extracted from the codestream and its entropy is calculated. The y-axis shows the average entropy value for each of the 23 JPEG2000 data-packets. The data-packets 1-8 are completely encrypted and data-packet 9 is partially encrypted.

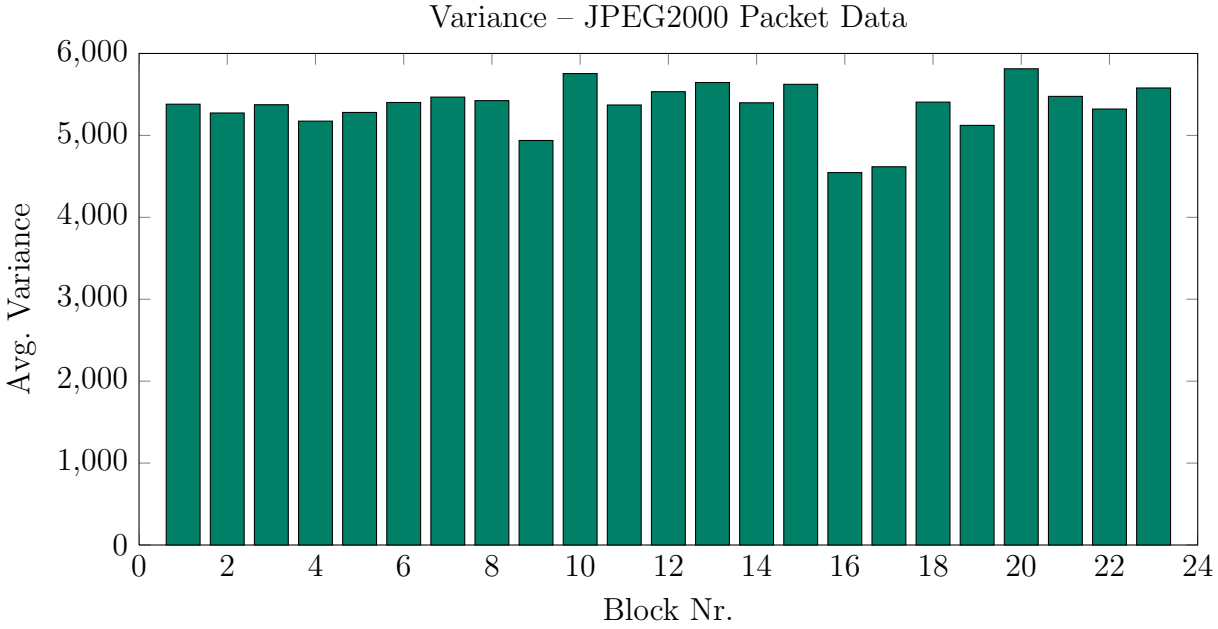


Figure D.4: The figure above shows the variance based on image cam1_1 from the SCFace image database [46]. Therefore, the image is converted into the JPEG2000 file-format. Afterwards each JPEG2000 data-packet is extracted from the codestream and its variance is calculated. The y-axis shows the average variance value for each of the 23 JPEG2000 data-packets. The data-packets 1-8 are completely encrypted and data-packet 9 is partially encrypted.

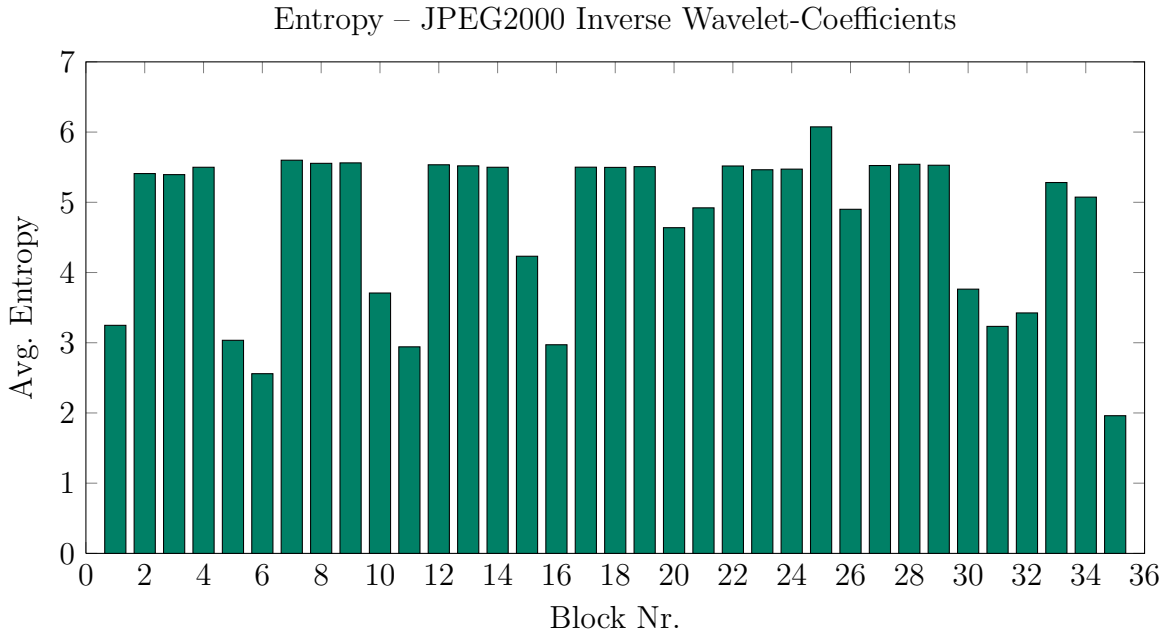


Figure D.5: The figure above shows the entropy based on image cam1_1 from the SCFace image database [46]. Therefore, the image is decoded until the JPEG2000 inverse Wavelet transformation has been computed. Afterwards the array of Wavelet-coefficients is partitioned into non-overlapping rectangular blocks, which are used to compute the entropy. The y-axis shows the average entropy for each image block. As evident from Figure 7.3 the following image blocks are unencrypted: 1, 5, 6, 10, 11, 15, 16, 20, 21, 25, 26, 30, 31, 32, 33, 34 and 35.

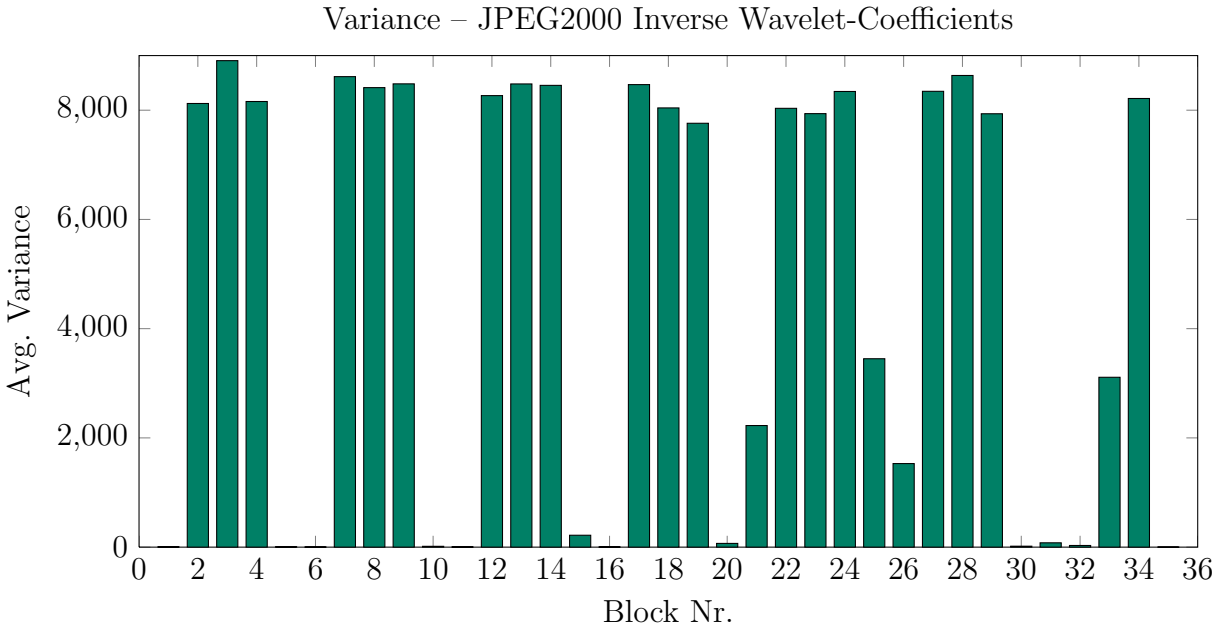


Figure D.6: The figure above shows the variance based on image cam1_1 from the SCFace image database [46]. Therefore, the image is decoded until the inverse Wavelet transformation has been computed. Afterwards the array of Wavelet-coefficients is partitioned into non-overlapping rectangular blocks, which are used to compute the variance. The y-axis shows the average variance for each image block. As evident from Figure 7.3 the following image blocks are unencrypted: 1, 5, 6, 10, 11, 15, 16, 20, 21, 25, 26, 30, 31, 32, 33, 34 and 35.

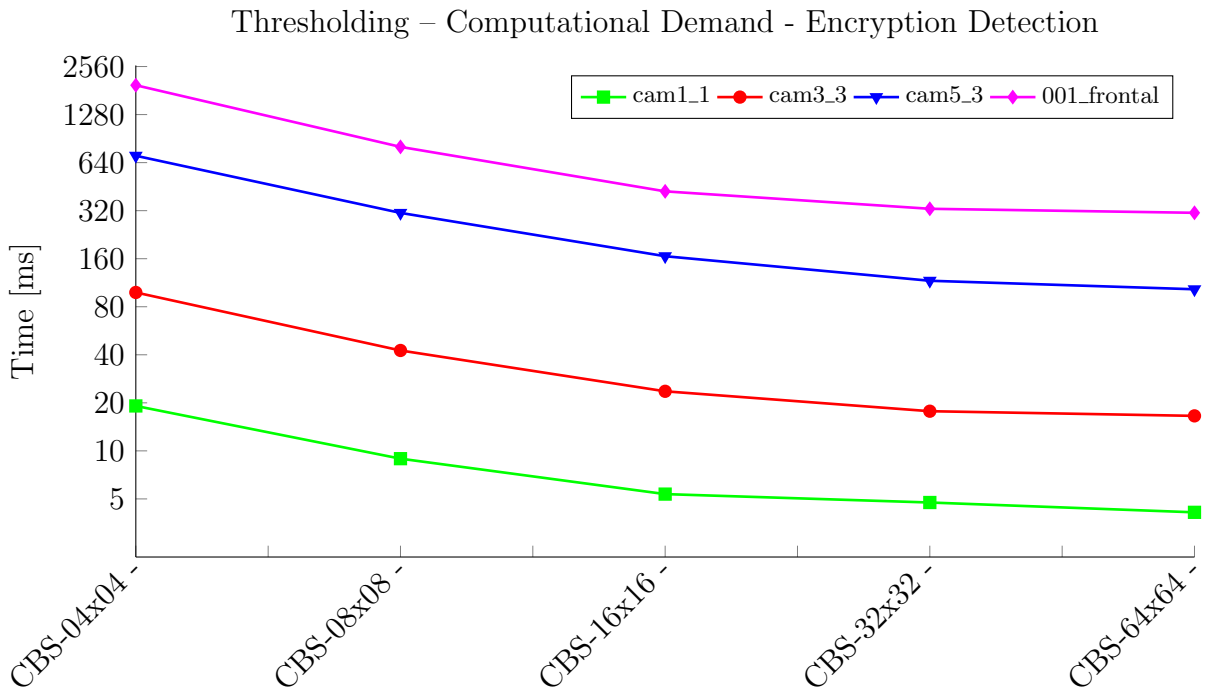


Figure D.7: This figure shows the computational demand, in milliseconds, required to detect the encrypted image blocks by the automated RoI detection method Thresholding. The y-axis shows the time, in milliseconds, required to detect the encrypted image blocks.

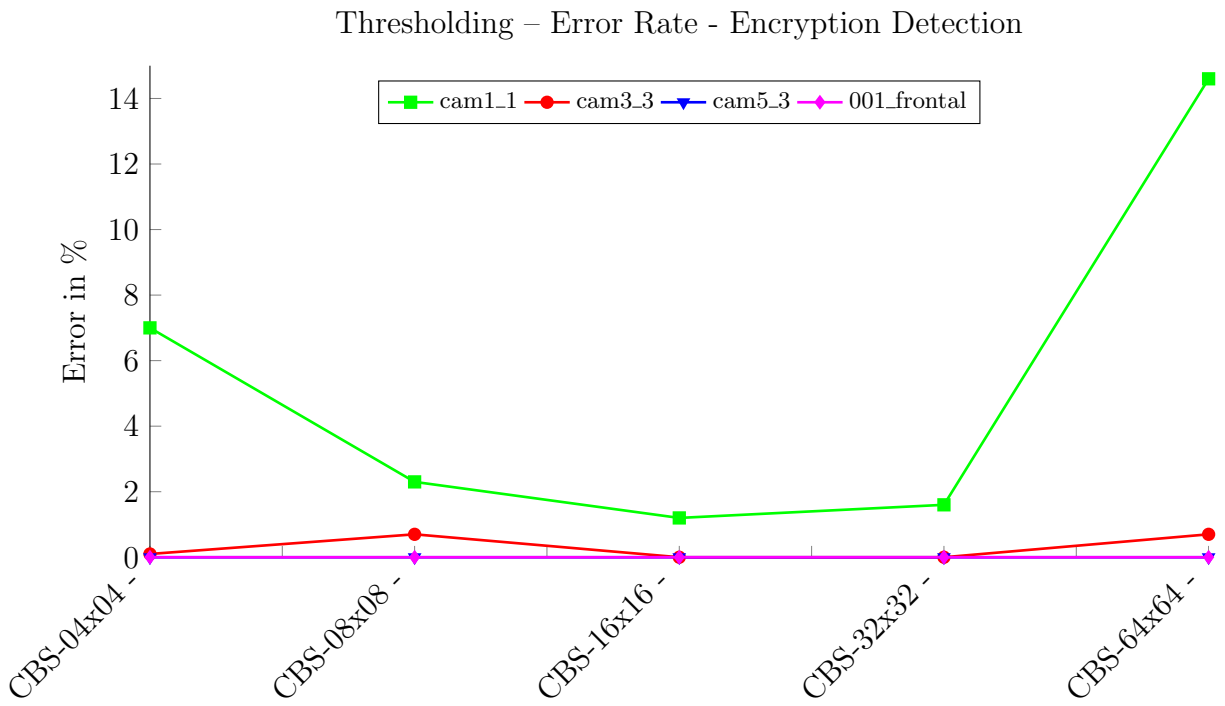


Figure D.8: The figure above shows the error rate caused by the automated RoI detection method Thresholding. The y-axis depicts the error rate, in percent, for detecting the encrypted RoI border wrongly.

Figure D.9: The figure above shows the error caused by the automated RoI detection method Thresholding. The y-axis depicts the wrongly detected encrypted RoI borders (out of 1000 simulations).

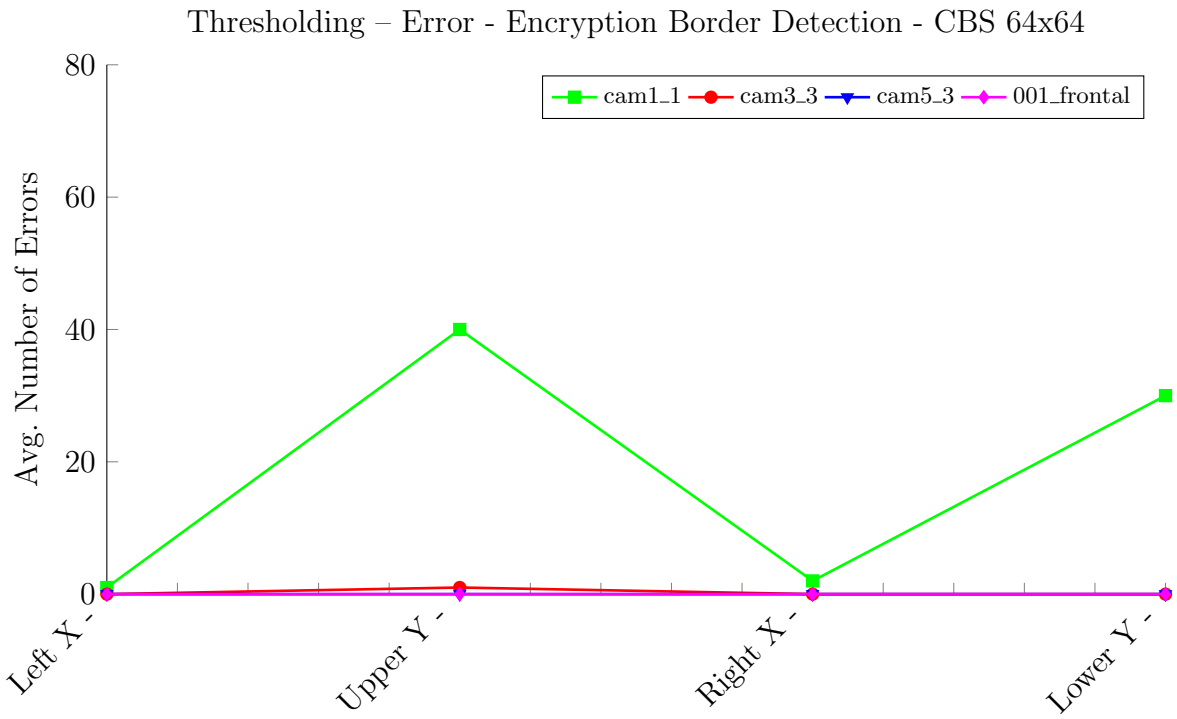


Figure D.10: The figure above shows the error caused by the automated RoI detection method Thresholding. The given results are based on a code-block-size of 4×4 pixel, used while encoding the JPEG2000 image. The y-axis depicts the wrongly detected encrypted RoI borders (out of 1000 simulations).

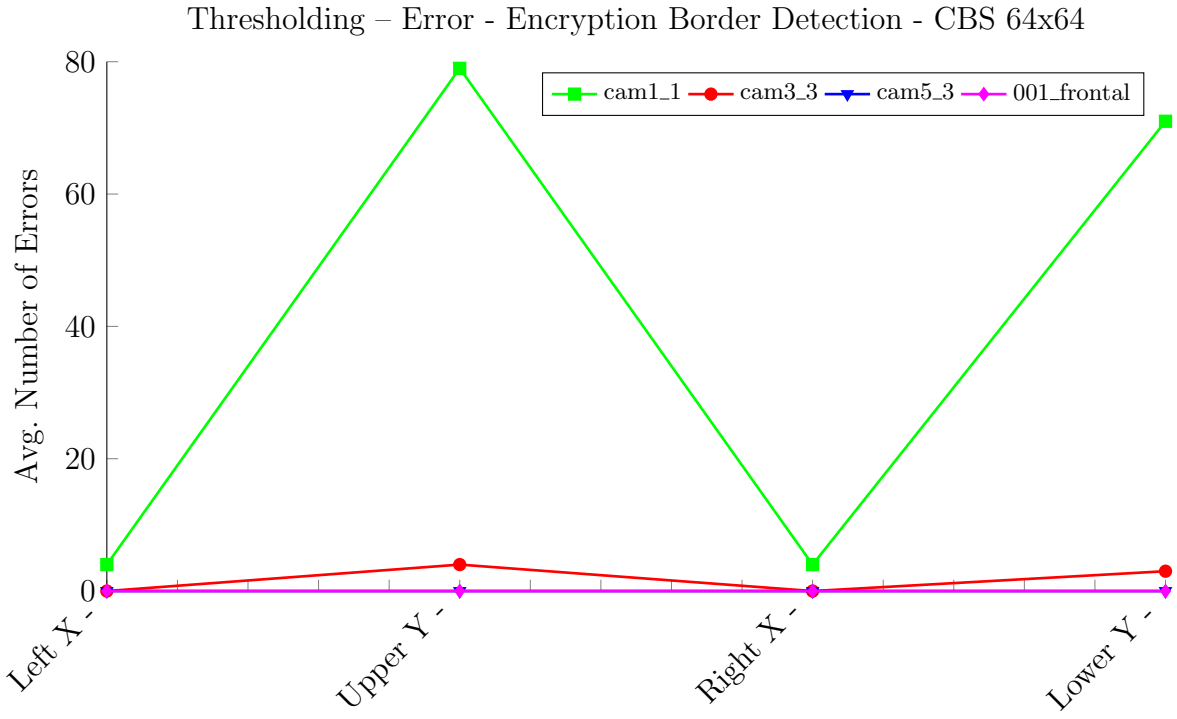


Figure D.11: The figure above shows the error caused by the automated RoI detection method Thresholding. The given results are based on a code-block-size of 64×64 pixel, used while encoding the JPEG2000 image. The y-axis depicts the wrongly detected encrypted RoI borders (out of 1000 simulations).

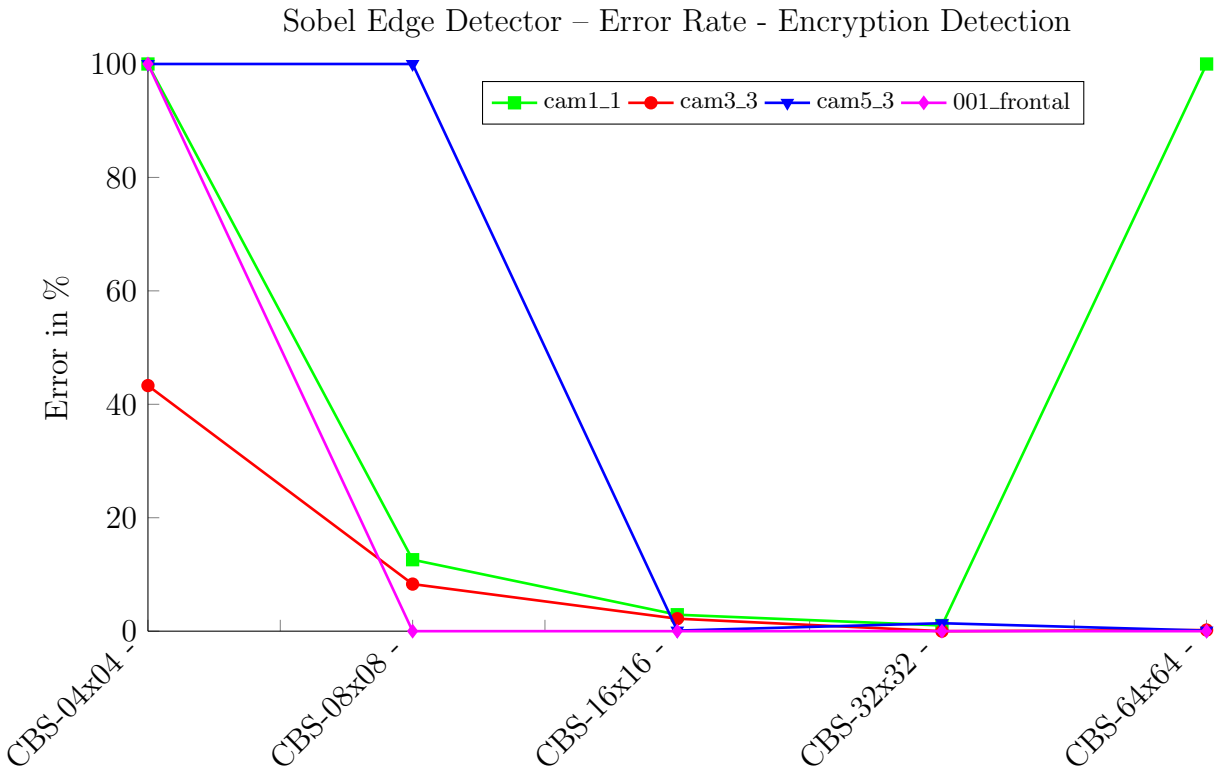


Figure D.12: The figure above shows the error, caused by applying the Sobel Edge Detector, to detect the encrypted image region. The y-axis shows the error rate logarithmically, for detecting the encrypted image regions correctly.

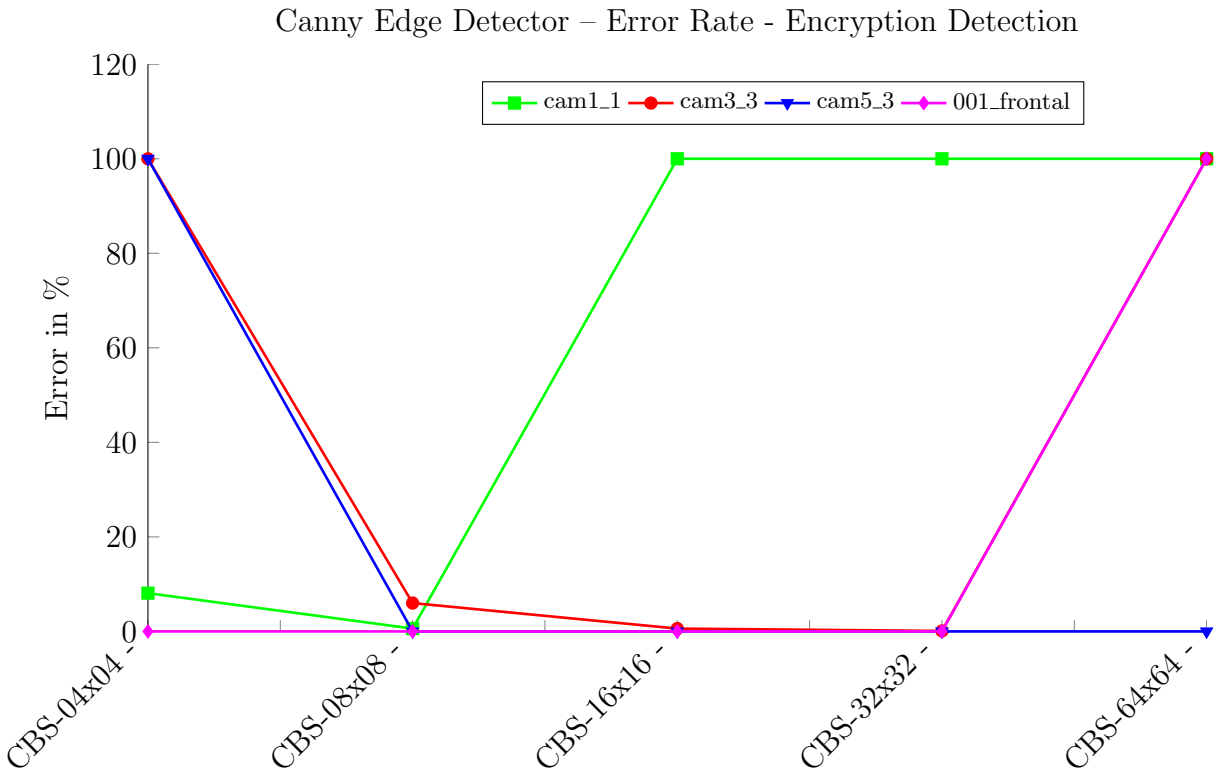


Figure D.13: This figure shows the error, caused by applying the Canny Edge Detector, to detect the encrypted image region. The y-axis shows the error rate logarithmically, for detecting the encrypted image regions correctly.

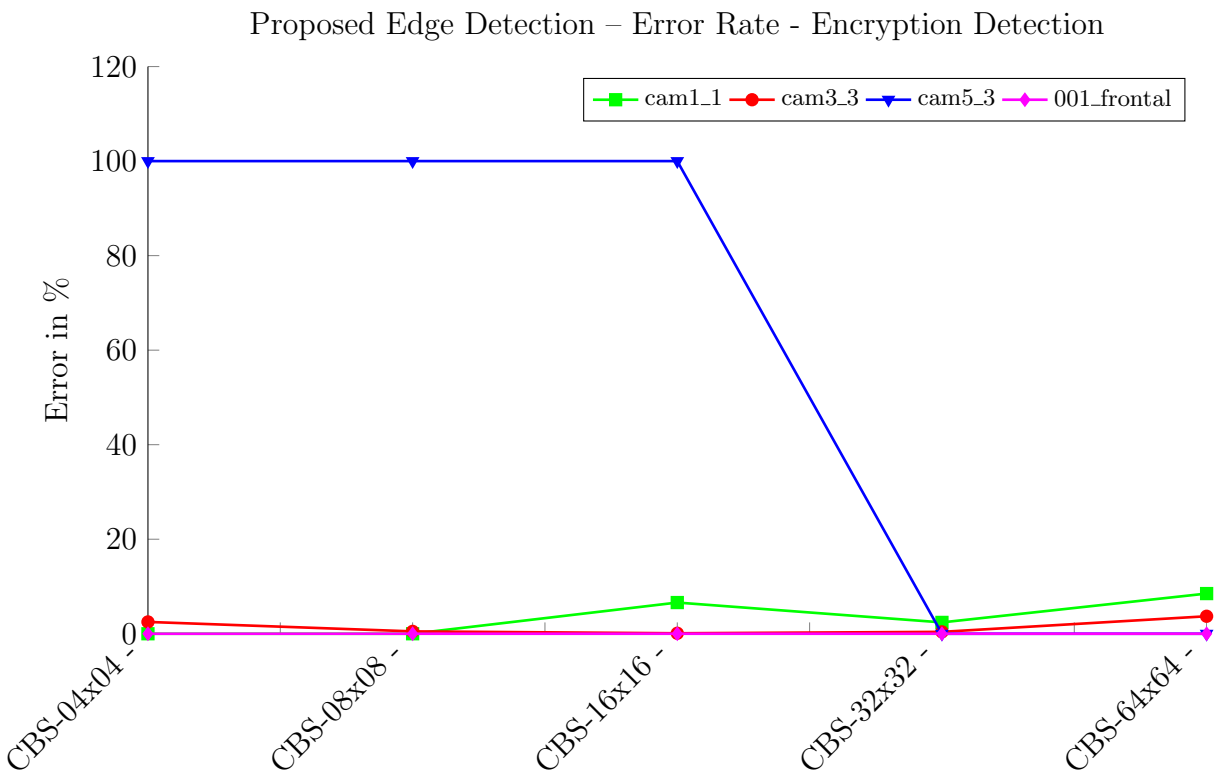


Figure D.14: This figure shows the error, caused by applying the Edge detection method proposed by us, to detect the encrypted image region. The y-axis shows the error logarithmically, for detecting the encrypted image region correctly

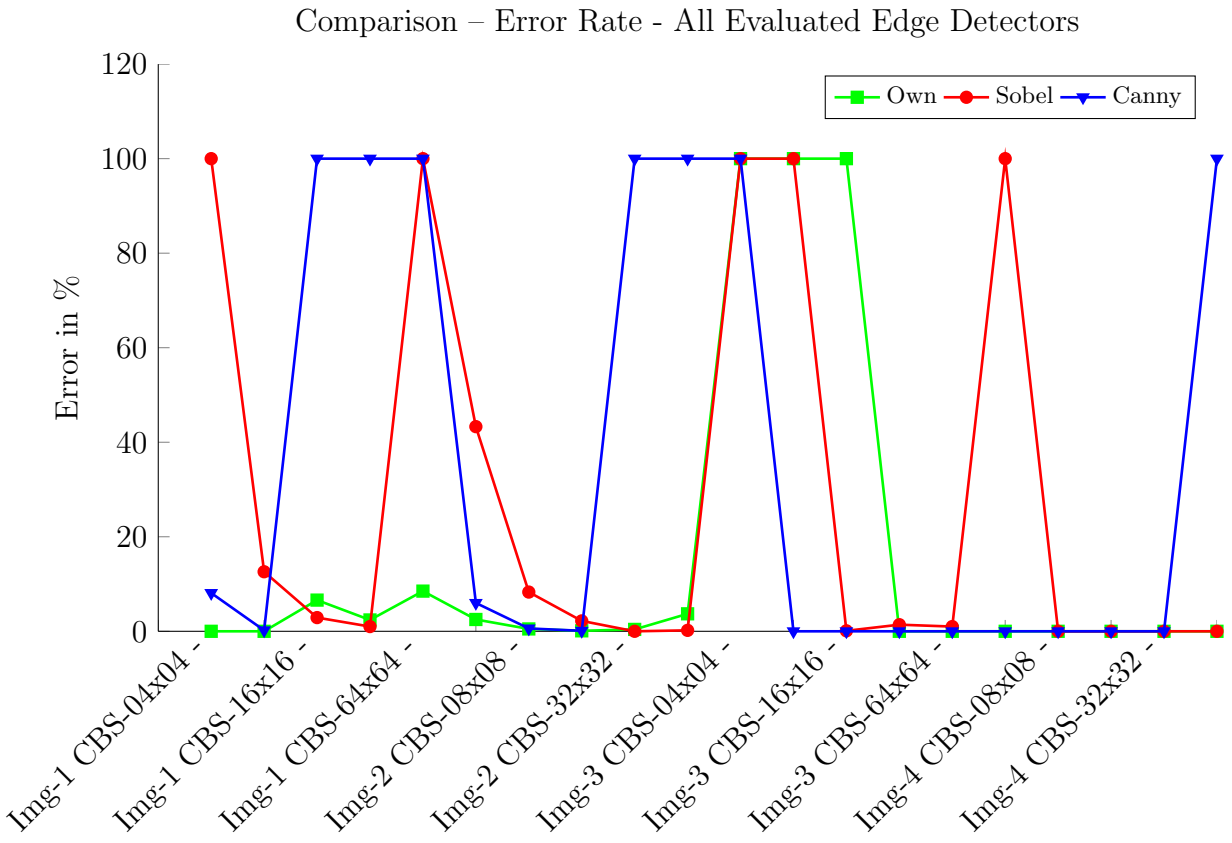


Figure D.15: This figure compares the error, caused by the edge detectors evaluated by this work. The y-axis shows the error rate for detecting the encrypted image region correctly.

Bibliography

[1] A. W. Senior, S. Pankanti, A. Hampapur, L. M. G. Brown, Y. li Tian, A. Ekin, J. H. Connell, C.-F. Shu, and M. Lu, "Enabling Video Privacy through Computer Vision," *IEEE Security & Privacy*, vol. 3, pp. 50–57, May 2005.

[2] W. H. Widen, "Smart Cameras and the Right to Privacy," *Proceedings of the IEEE*, vol. 96, pp. 1688–1697, October 2008.

[3] F. Dufaux, M. Ouaret, Y. Abdeljaoued, A. Navarro, F. Vergnenegre, and T. Ebrahimi, "Privacy Enabling Technology for Video Surveillance," in *SPIE Mobile Multimedia/Image Processing for Military and Security Applications*, Lecture Notes in Computer Science, IEEE, 2006.

[4] D. Engel, T. Stütz, and A. Uhl, "A survey on JPEG2000 encryption," *Multimedia Systems*, vol. 15, pp. 243–270, August 2009.

[5] C. Stubhann, "Selektive Verschlüsselung einer Region-of-Interest in JPEG2000-Bitströmen," Master's thesis, University of Applied Sciencies Salzburg, August 2012.

[6] M. J. Gormish, D. Lee, and M. W. Marcellin, "JPEG2000: overview, architecture and applications," in *InProceedings of ICIP'2000*, vol. 2, (Menlo Park, CA, USA), pp. 29–32, Ricoh California Research Center, September 2000.

[7] K. L. Gray, "The JPEG2000 Standard," *Technical University Munich, Institute for Communication Networks*, February 2003.

[8] M. D. Adams, "The JPEG-2000 Still image compression standard." Department of Electricaland Computer Engineering, University of Victoria, December 2005.

[9] D. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Norwell, MA, USA: Kluwer Academic Publishers, November 2001.

[10] V. Sanchez and A. Basu, "The JPEG2000 Image Compression Standard." http://webdocs.cs.ualberta.ca/~anup/Courses/604/NOTES/slide_jpeg2000.pdf, February 2003. Accessed: 15/02/2013.

[11] intoPIX, "Everything you always wanted to know about JPEG2000." http://www.intopix.com/pdf/JPEG%202000%20Handbook.pdf, 2008. Accessed: 15/02/2013.

[12] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: an overview," *Consumer Electronics, IEEE Transactions on*, vol. 46, pp. 1103–1127, November 2000.

[13] V. M. Potdar, S. Han, and E. Chang, "A survey of digital image watermarking techniques," in *Industrial Informatics, 2005. INDIN '05. 2005 3rd IEEE International Conference on*, pp. 709–716, August 2005.

[14] M. Kharrazi, H. T. Sencar, and N. Memon, "Image Steganography: Concepts and Practice," *Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore*, April 2004.

[15] B. Green, "Edge Detection Tutorial." http://dasl.mem.drexel.edu/alumni/bGreen/, 2002. Accessed: 04/03/2013.

[16] F. Dufaux and T. Ebrahimi, "Scrambling for Privacy Protection in Video Surveillance Systems," in *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 18, pp. 1168–1174, 2008.

[17] Y. Kim, S. Jin, and Y. Ro, "Scalable Security and Conditional Access Control for Multiple Regions of Interest in Scalable Video Coding," in *Digital Watermarking, 6th International Workshop, IWDW 2007* (Y. Shi, H.-J. Kim, and S. Katzenbeisser, eds.), vol. 5041 of *Lecture Notes in Computer Science*, pp. 71–86, Springer Berlin Heidelberg, December 2007.

[18] D. S. Taubman and M. W. Marcellin, "JPEG2000: standard for interactive imaging," *Proceedings of the IEEE*, vol. 90, pp. 1336–1357, August 2002.

[19] J. Hämmerle-Uhl, R. Schraml, and A. Uhl, "Privacy Enhancing Technologies in Video Surveillance applied to JPEG2000 Codestreams," *In Proceedings of the IEEE International Workshop on Multimedia Signal Processing (MMSP'12)*, pp. 95–100, September 2012.

[20] P. Wayner, *Disappearing Cryptography: Information Hiding: Steganography & Watermarking*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 3 ed., January 2009.

[21] L. Zhang, H. Wang, and R. Wu, "A high-capacity steganography scheme for JPEG2000 baseline system," *Image Processing, IEEE Transactions on*, vol. 18, pp. 1797–1803, August 2009.

[22] H. Gao, G. Liu, X. Li, and Y. Xu, "A Robust Watermark Algorithm for JPEG2000 Images," in *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security*, vol. 2 of *IAS '09*, (Washington, DC, USA), pp. 230–233, IEEE Computer Society, August 2009.

[23] H. Ming-Shing, T. Din-Chang, and H. Yong-Huai, "Hiding digital watermarks using multiresolution wavelet transform ," *Industrial Electronics, IEEE Transactions on*, vol. 48, pp. 875–882, October 2001.

[24] K. A. Navasa, S. Nithya, R. Rakhi, and M. Sasikumar, "Lossless Watermarking in JPEG2000 for EPR Data Hiding," *Proceedings IEEE EIT*, pp. 697–702, June 2007.

[25] P. Meerwald, "Quantization Watermarking In The JPEG2000 Coding Pipeline," in *In 5th International Working Conference on Communication and Multimedia Security* (J. D. Ralf Steinmetz and M. Steinebach, eds.), (Darmstadt, Germany), pp. 69–79, Kluver Academic Publishing, May 2001.

[26] A. Katsutoshi, K. Hiroyuki, and K. Hitoshi, "A Method for Embedding Binary Data into JPEG2000 Bit Streams Based on the Layer Structure," *Technical report of IEICE. DSP*, vol. 101, pp. 17–24, October 2001.

[27] F. Dufaux and T. Ebrahimi, "Smart video surveillance system preserving privacy," *Proceedings of SPIE*, vol. 5685, pp. 54–63, April 2005.

[28] "Joint Photographic Expert Group (JPEG)." http://www.jpeg.org/jpeg2000/. Accessed: 13/02/2013.

[29] A. N. Skodras, C. A. Christopoulos, and T. Ebrahimi, "JPEG2000: the upcoming still image compression standard," *Pattern Recognition Letters*, vol. 22, pp. 1337–1345, October 2001.

[30] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, pp. 30–44, April 1991.

[31] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. Norwell, MA, USA: Kluwer Academic Publishers, 1st ed., 1992.

[32] T. Acharya and P. Tsai, *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. Hoboken, New Jersey: John Wiley & Sons, 1 ed., October 2004.

[33] C. Christopoulos, J. Askelof, and M. Larsson, "Efficient methods for encoding regions of interest in the upcoming JPEG2000 still image coding standard," *Signal Processing Letters, IEEE*, vol. 7, pp. 247–249, September 2000.

[34] D. Engel, T. Stütz, and A. Uhl, "Format-compliant JPEG2000 encryption in JPSEC: Security, Applicability, and the Impact of Compression Parameters," *EURASIP Journal on Information Security*, vol. 2007, pp. 8:1–8:20, January 2007.

[35] J. Apostolopoulos, S. Wee, F. Dufaux, T. Ebrahimi, Q. Sun, and Z. Zhang, "The emerging JPEG-2000 security (JPSEC) standard," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, Lecture Notes in Computer Science, pp. 4 pp. –3885, IEEE, May 2006.

[36] Q. Sun and Z. Zhishou, "JPSEC: Security Part of JPEG2000 Standard," *Information Technology Standards Committee*, pp. 21–30, October 2006.

[37] F. Dufaux and T. Ebrahimi, "Region-Based Transform-Domain Video Scrambling," in *SPIE Proceeding Visual Communications and Image Processing*, Lecture Notes in Computer Science, IEEE, 2006.

[38] S.-C. Cheung, J. Paruchuri, and T. Nguyen, "Managing privacy data in pervasive camera networks," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pp. 1676 –1679, October 2008.

[39] A. Uhl and R. Schraml, "J2K Privacy," *Department of Computer Sciences University of Salzburg, Austria*, 2012.

[40] Y. Mao and W. Min, "A joint signal processing and cryptographic approach to multimedia encryption," *Image Processing, IEEE Transactions on*, vol. 15, no. 7, pp. 2061–2075, 2006.

[41] T. Stütz and A. Uhl, "On format-compliant iterative encryption of JPEG2000," in *in Proceedings of the Eighth IEEE International Symposium on Multimedia (ISM06), (Los Alamitos*, (San Diego, CA, USA), pp. 985–990, IEEE Computer Society, December 2006.

[42] T. Stütz and A. Uhl, "On efficient transparent jpeg2000 encryption," in *Proceedings of the 9th workshop on Multimedia & security*, (Dallas, Texas, USA), pp. 97–108, Association for Computing Machinery (ACM), September 2007.

[43] T. Köckerbauer, M. Polak, T. Stütz, and A. Uhl, "GVid - Video Coding and Encryption for Advanced Grid Visualization," in *Proceedings of the 1st Austrian Grid*

Symposium (J. Volkert, T. Fahringer, D. Kranzlmüller, and S. W., eds.), vol. 210, (Schloss Hagenberg, Austria), pp. 204–218, Austrian Computer Society, OCG Verlag, December 2005.

[44] H. Hofbauer, C. Rathgeb, A. Uhl, and P. Wild, "Image metric-based biometric comparators: A supplement to feature vector-based Hamming distance?," in *Biometrics Special Interest Group (BIOSIG), 2012 BIOSIG - Proceedings of the International Conference of the*, (Salzburg, Austria), pp. 1–5, Deptartment of Comput. Science, University of Salzburg, September 2012.

[45] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, 2004.

[46] M. Grgic, K. Delac, and S. Grgic, "SCface — surveillance cameras face database," *Multimedia Tools Appl.*, vol. 51, pp. 863–879, February 2011.

[47] Y. Wu and R. Deng, "Compliant encryption of JPEG2000 codestreams," in *Image Processing, 2004. ICIP '04. 2004 International Conference on*, vol. 5, pp. 3439–3442, IEEE, October 2004.

[48] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*. Multimedia Information and Systems, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2 ed., November 2007.

[49] F. Petitcolas, R. Anderson, and M. Kuhn, "Information hiding a survey," *Proceedings of the IEEE*, vol. 87, pp. 1062 –1078, July 1999.

[50] B. Vassiliadis, V. Fotopoulos, A. Ilias, and A. Skodras, "Protecting Intellectual Property Rights and the JPEG2000 Coding Standard," in *Panhellenic Conference on Informatics* (P. Bozanis and E. Houstis, eds.), vol. 3746 of *Lecture Notes in Computer Science*, pp. 705–715, Springer Berlin Heidelberg, November 2005.

[51] I. Cox, M. Miller, J. Bloom, and C. Honsinger, "Digital Watermarking," *Journal of Electronic Imaging*, vol. 11, no. 3, pp. 414–414, 2002.

[52] I. J. Cox, T. Kalker, and H.-K. Lee, eds., *Digital Watermarking, Third International-Workshop, IWDW 2004, Seoul, SouthKorea, October 30 - November 1, 2004, Revised Selected Papers*, vol. 3304 of *Lecture Notes in Computer Science*. Seoul, Sourth Korea: Springer, 2005.

[53] G. Saraswat Pradeep Kumar and R. K., "Digital Image Steganography - A Gentle Overview," *VSRD International Journal of Computer Science & Information Technology*, vol. 2, pp. 129–136, February 2012.

[54] P. Su, *Information hiding in digital images: watermarking and steganography*. PhD thesis, Los Angeles, CA, USA, 2003. AAI3103970.

[55] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in *Advances in Cryptology – CRYPTO˜'83*, pp. 51–67, Plenum, 1984.

[56] T. Morkel, J. Eloff, and M. Olivier, "An Overview of Image Steganography," in *Proceedings of the Fifth Annual Information Security South Africa Conference (ISSA2005)* (H. Henter, J. Eloff, L. Labuschagne, and M. Eloff, eds.), (Sandton, South Africa), July 2005.

[57] "ISO/IEC 15444-1. Information Technology - JPEG2000 Image Coding System, Part 1: Core Coding System," December 2000.

[58] D. Engel, A. Uhl, and A. Unterweger, "Region of Interest Signalling For Encrypted JPEG Images," 2013.

[59] C. E. Shannon, *A Mathematical Theory of Communication*. CSLI Publications, 1948.

[60] J. T. Inder, "Generalized Information Measures and Their Applications." http://www.mtm.ufsc.br/˜taneja/book/book.html, June 2001. Accessed: 03/03/2013.

[61] A. Lesne, "Shannon entropy: a rigorous mathematical notion at the crossroads between probability, information theory, dynamical systems and statistical physcis," *Mathematical Structures in Computer Science*, vol. 1, pp. 1–43, January 2013.

[62] Y. Wu, Y. Zhou, G. Saveriades, S. Agaian, J. P. Noonan, and P. Natarajan, "Local Shannon entropy measure with statistical tests for image randomness," *Information Sciences*, vol. 222, pp. 323–342, February 2013.

[63] T. Arens, F. Hettlich, C. Karpfinger, U. Kockelkorn, K. Lichtenegger, and H. Stachel, *Mathematik*. Spektrum Akademischer Verlag, 2 ed., October 2011.

[64] R. Maini and H. Aggarwal, "Study and Comparison of Various Image Edge Detection Techniques," *International Journal of Image Processing*, vol. 3, no. 1, pp. 1–12, 2010.

[65] C. Philippe, "Image segmentation - introduction to signal and image processing," *University of Basel*, p. 193, May 2012.

[66] J. Canny, "A Computational Approach to Edge Detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 8, pp. 679–698, June 1986.