Cite this article as:
Poljicak, A., Botella, G., Garcia, C. et al. J Real-Time Image Proc (2016). doi:10.1007/s11554-016-0616-9

Ante Poljicak · Guillermo Botella · Carlos Garcia
Luka Kedmenec · Manuel Prieto-Matias

Portable Real-Time DCT Based Steganography Using OpenCL

First online: 05 July 201

Abstract In this paper a steganographic method for real time data hiding is proposed. The main goal of the research is to develop steganographic method with increased robustness to unintentional image processing attacks. In addition, we prove the validity of the method in real time applications. The method is based on a discrete cosine transform (DCT) where the values of a DCT coefficients are modified in order to hide data. This modification is invisible to a human observer. We further the investigation by implementing the proposed method using different target architectures and analyse their performance. Results show that the proposed method is very robust to image compression, scaling and blurring. In addition, modification of the image is imperceptible even though the number of embedded bits is high. The steganalysis of the method shows that the detection of the modification of the image is unreliable for a lower relative payload size embedded. Analysis of OpenCL implementation of the proposed method on four different target architectures shows considerable speedups.

Keywords Steganography, Real-time, OpenCL, GPU, Parallel Processing

This work has been supported by the Spanish Ministry of Economy and Competitiveness (MINECO) through the research projects TIN 2015-65277-R and TIN 2012-32180.

Ante Poljicak
University of Zagreb, Croatia
Tel.: +385-1-2371080
E-mail: ante.poljicak@grf.hr

Guillermo Botella
Complutense University of Madrid, Spain
E-mail: gbotella@ucm.es

Carlos Garcia
Complutense University of Madrid, Spain
E-mail: garsanca@ucm.es

Luka Kedmenec
University of Zagreb, Croatia

Manuel Prieto-Matías
Complutense University of Madrid, Spain
E-mail: mpmatias@ucm.es

1 Introduction

Numerous ways of communication in our networked society enables easy and instant sharing of digital images. To a photographer, this presents an opportunity for a promotion and a way to do business. However, it also presents a problem since it is very easy for someone to reshare the images and violate copyrights of the original owner. To mitigate this problem there are various watermarking methods for the protection of the ownership of digital images. Instant sharing of digital images also enables a covert communication by means of steganography. Both, watermarking and steganography hide additional information in an image and, Even though they are closely related and have many common traits there is one fundamental difference. Steganographic methods are those in which data payload hidden in an image is unrelated to the image itself, while watermarking methods are those which hide data payload that is, on one way or another, related to the image itself [8].

One approach to hide data in an image is to exploit the image format. For example, an Exchangeable image file (EXIF) header can be modified in order to hide a secret message as suggested by [1]. The same can be done with JPEG headers [16]. This approach of data hiding is easy to implement, and the image information is usually unaltered i.e. no image degradation is introduced during encoding. Unfortunately the alternation of data in the image format header is easy to detect, therefore, hidden data can easily be extracted, deleted or altered.

Other approach is to hide data by the manipulation of image pixel values in spatial domain that are undetectable by HVS (Human Visual System). One of the earliest methods was proposed by [18] where the data is embedded in four LSBs (Least significant bits) of an image. The idea of LSB manipulation is very simple. LSB of each pixel is overwritten with the information to be hidden. This approach introduces some degradation but it is imperceivable for a human observer as long as only least significant bits are modified. Other advantage is that the

LSB methods are usually simple and have a large capacity. In addition, overall security of a LSB method can be improved by using existing cryptographic methods (e.g. [25, 20]) to encrypt data before hiding. Unfortunately, LSB methods are not robust because even basic image processing operations such as compression, bluring or unsharp masking destroy the information hidden in LSBs. There are also more modern and more robust LSB based methods as those proposed by [6, 10, 4]. These methods are more robust, however, this increase in robustness is followed by a greater complexity of the method and a lower capacity.

Third possible approach is to embed the information in an image by manipulation of the image coefficients in a frequency domain. To hide data payload, an image is first transformed using a transform such as DCT (Discrete Cosine Transform), DWT (Discrete Wavelet Transform) or DFT (Discrete Fourier Transform), after which its coefficients are altered. Since different transforms have distinct properties, a steganography methods based on different transforms will also have distinct properties regarding the robustness against different image processing attacks, capacity and perceptibility. Methods based on DCT such as those proposed in [21, 13] are known to have a high robustness to DCT based compression methods [7]. On the other hand DWT methods proposed in [19, 5] show excellent spatial and frequency localization which is useful for modeling of HVS. Methods based on DFT due to its native invariance to translation and rotation show a high robustness to geometrical attacks and complex attacks such print and scan (PS) process as reported in [14, 23].

All steganographic methods are constrained by three main properties. Robustness to attacks, capacity of hidden data, and perceptibility of hidden data. Unfortunately, it is not possible for the steganographic method to simultaneously be very robust, have a large capacity and the low perceptibility of hidden data. Therefore, during the design of a data hiding method, the requirements on robustness, capacity and perceptibility is determined by the future application of the method. In this paper we propose a data hiding method for general use, with focus on social networks. The method is based on DCT because the two most common image processing attacks in general are resizing and JPEG compression. As already mentioned, methods based on DCT have the high robustness to JPEG compression.

Regarding the hardware platform used, the portable industry in recent years has made many efforts in reducing its cost. However, one of the main challenges is to achieve a unified programming way for heterogeneous devices such as embedded systems, mobile-devices, desktop computers and high performance systems. OpenCL [12] is a novel paradigm which reflects all these efforts. OpenCL does not only allow portability but also expresses parallel programming which increases performance rates.

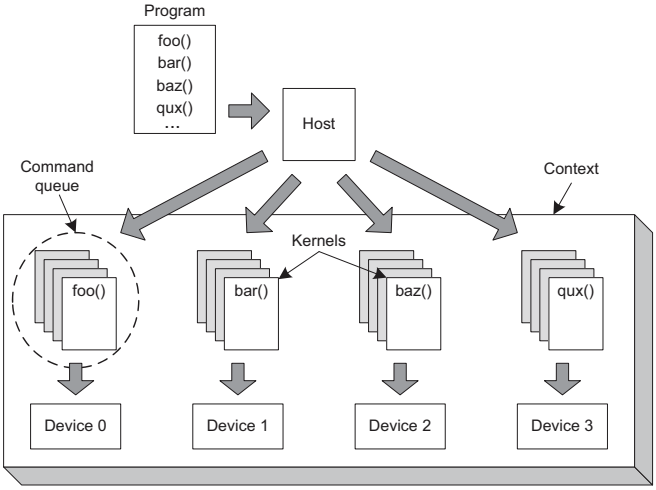

Fig. 1 OpenCL Parallel Programing Paradigm. Source [24]

This paper is organized as follows. In section 2 an overview of the OpenCL paradigm is given. The proposed steganographic method is explained in section 3. The detailed analysis of the method is given in section 4. The hardware implementation of the method is considered in section 5 and section 6. Results of the hardware implementation are discussed in section 7. In section 8 conclusion and future research plans are given.

2 OpenCL

As mentioned previosly, OpenCL paradigm has been chosen to accelerate the program written in C to get better performance and faster execution. OpenCL is a platform independent framework for description and execution of distributed algorithms. An OpenCL scheme is based on a host-device model, where the host manages several connected devices and it is in charge of the kernels execution.

2.1 Platform and Memory Model

The platform model defines physical components of the OpenCL capable hardware and how they interact with the host. The system can contain several independent computing devices, each one connected to the host. Each device has one or more computing units (CUs) which are not directly connected to each other. Computation is performed by processing elements (PEs), which can communicate and synchronize with others CUs present in the same PE.

PEs have access to different memories. On one hand, we have a large *global* and *constant* memory which are accessible from all PEs. On the other hand, PEs of the same unit share a smaller amount of a *local* memory and each PE has also its own *private* memory. The memory

type is assigned by a programmer using the attributes global, constant, local and private on declaration.

The platform model realization depends on the specific hardware. For instance, focusing on a multicore platform, the devices correspond with the physical cores. PEs are realized by kernel engines.

2.2 Execution Model

The OpenCL execution model defines how the problem is mapped to a hardware. It divides the problem thanks to separate *work-groups*, consisting of multiple *work-items*. *Work-items* of the same *work-group* can communicate and synchronize between them. In contrast, separate *work-groups* are independent.

OpenCL provides kernel built-in functions to get the current global and local sizes as well as identifiers (IDs) in order to differentiate between *work-items* depending on their rank in the work space.

Local ID matches an items ID within its *work-group*. Moreover, synchronization and math functions are available. Usually (depending on the specific application), there are more *work-items* than kernel engines. The scheduler supports a list of every *work-item* for each *work-group*, which have to be run and which is currently on-air. A kernel engine does not only execute a single *work-item* at a given time, but also executes multiple *work-items* with different progress. This is so-called pipeline parallelism, which is ever-present in the hardware design and allows to simultaneously execute multiple items within a single engine. The fact that the local memory is only valid within groups implies that only items of the same group are executed in parallel using one kernel engine.

Each kernel is specially coded and can be executed in one or more compliant devices. Kernels are sent to a device or devices by a host application, which is the basic C/C++ application that runs on the host. For example, host can send kernels to a single device as the GPU, but also the same CPU can act as the device executing kernels. To manage connected devices, host applications use a container called a context as it is shown in Fig. 1 with the command queue associated to each device. In general this works straightforward, from a kernel container called a program, host selects a function to create a kernel, and then it associates the kernel with argument data and sends it to a command queue. This is the mechanism through which the host tells devices what to do. Once the kernel is enqueued, the device can execute this particular function. In this way, applications can configure different devices to execute different tasks and every task can operate on different data.

2.3 Kernel Distribution

OpenCL permits two ways to distribute kernels. Most implementations allow or enforce the compilation by the host just before the kernel is executed. Therefore, kernels must be available in the source to be device independent, which is not preferred for commercial applications. Already compiled kernels, however, depend on the hardware they were built for, but the kernel can be distributed as binary, and the host doesn't need to be powerful to compile the kernel, which is especially beneficial for low-power processors. To fill the gap, Standard Portable Intermediate Representation (SPIR) is used to define a representation of the kernel on a lower logical level, which still allows a target-dependent optimization. SPIR is already supported by several drivers, e.g. Intel or AMD.

3 Proposed method

The proposed method is based on our preliminary work published in [15]. The method belongs to the category of blind data hiding techniques. Therefore, in order to extract the hidden information, the original image is not needed at the decoder. A cover image is first transformed into YCbCr color space in order to separate the lightness and chromaticity information of the image. A data payload, unrelated to the image, is embedded into DCT coefficients of the Y component (lightness) of an image. The security of the hidden information is ensured by the random distribution of bits which is modulated with a secret key.

3.1 Encoder

The block diagram of the encoder is shown in Fig. 2. The cover image $I(x, y, z)$ is transformed to YCbCr color space. Lightness component $Y(x, y)$ of the image is then separated into 8×8 blocks which are then transformed to the frequency domain using DCT. To hide the data payload, coefficients of the 8×8 blocks are modulated using (1). Note that only some blocks are modified. The total number of modified blocks depends on the number of the bits of the hidden message. Which blocks will be modified is determined by a pseudorandom number generator controlled with the secret key.

Within a 8×8 block there are 64 DCT coefficients, and only j-th coefficient in the block is substituted by a new value $M'_{j,k}$, which is calculated by multiplying the implementation factor α, a normalized mean value of all 64 coefficients in the block k and with 1 or -1 depending on the bit value of the data payload to be hidden (1). The factor of implementation α controls the strength of the implementation of the hidden information. Higher implementation factor results in more robustness; however, it also introduces more degradation to the stego image.

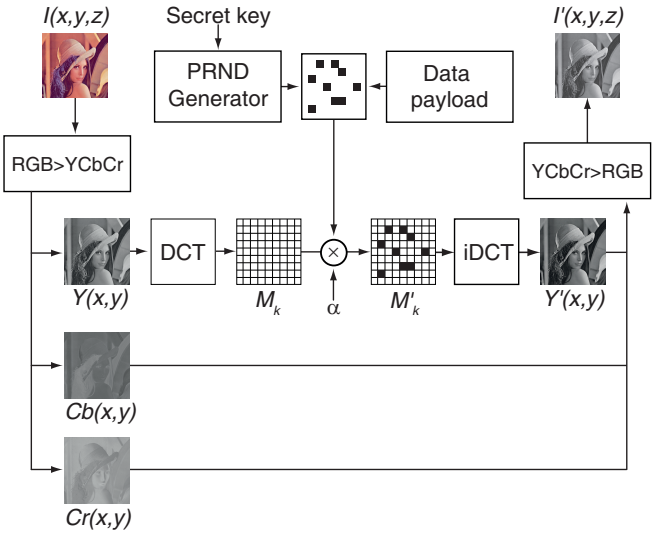

Fig. 2 Block diagram of the encoder.

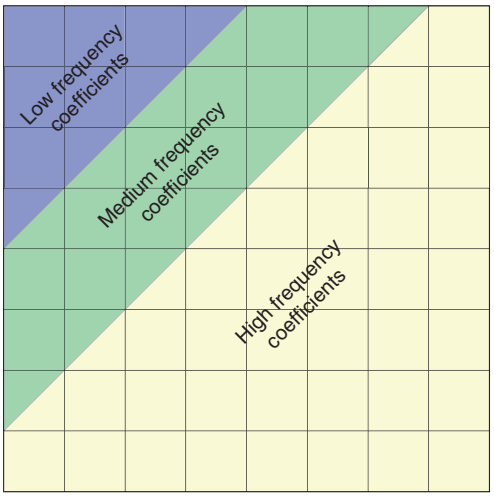

Fig. 3 Low, medium and high frequency DCT coefficents in an 8×8 block.

$$M'_{j,k} = \frac{\alpha W_k}{64} \sum_{i=1}^{64} M_i \tag{1}$$

Where $M'_{j,k}$ is a modified coefficient j in a block k, M_i denotes original coefficients in the block k, W_k denotes the k-th bit of the data payload, and α denotes the implementation factor.

As already mentioned, which blocks will be used to hide the information is determined by a secret key as a seed for a pseudorandom number generator. Therefore, each bit of the data payload is pseudo randomly distributed across the stego image. An 8×8 block has 64 coefficients which can be modified to embed one bit of the data payload. However, not all coefficients are equal in terms of the robustness or perceptibility [8]. Lower frequency coefficients (Fig. 3) are perceptually significant and robust to attacks. In other words, modifying low frequency coefficients will produce stronger degradation of the stego image, but the hidden message will be more robust against image processing attacks. In contrast, modification of higher frequency coefficients lead to less degradation of the stego image, but also to lower resistance against attacks. An example of the stego image, generated by the proposed method, and its cover image are shown in Fig. 4

(a)

(b)

Fig. 4 Example of the data payload implementation. ($\alpha = 1$, Image size: 1024×768 pixels); (a) Cover image, (b) Stego image with 2500 hidden bits (SSIM = 0.9989).

3.2 Decoder

Since the proposed method is blind, in order to extract the data payload at the decoder (Fig. 5) only stego image $I'(x, y, z)$ and a secret key are needed. Without the key, it is not possible to know the exact position of the blocks that contain hidden information. In addition, since the data payload is randomly distributed throughout the stego image, the key is used to get the exact order of the

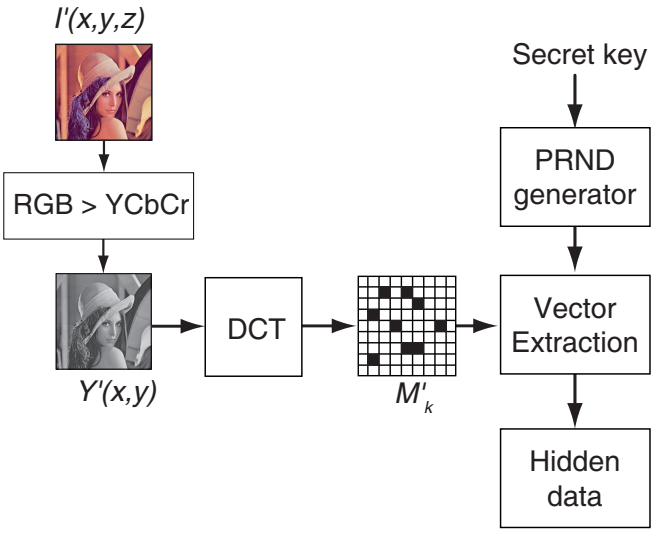

Fig. 5 Block diagram of the decoder.

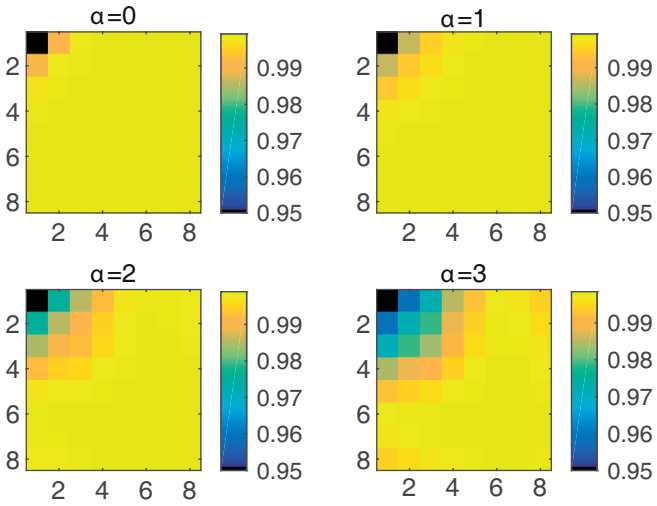

Fig. 6 Impact of the position of a modified DCT coefficient on the quality of the stego image for given implementation factor α. SSIM index gets larger as the frequency of the modified DCT coefficient is higher.

hidden bits to correctly retrieve the embedded data payload. Before the extraction of the hidden data, the stego image is converted to YCbCr color space. 8×8 blocks of the lightness component $Y'(x, y)$ is then transformed to the cosine domain to determine DCT coefficients.

By using a pseudorandom sequence generated by the secret key, the decoder extracts the values of the coefficients in particular blocks. The hidden data payload is reconstructed using the step unit function (2). For positive value of the coefficient recovered bit is 1, for negative value of the coefficient, recovered bit is 0. This simple approach ensures very fast processing at the decoder, and offers the increased robustness against attacks.

$$H = \begin{cases} 0, & \text{for } k < 0 \\ 1, & \text{for } k \geq 0 \end{cases} \tag{2}$$

4 Testing of the method

To test the method, the image database of 140 images was used. The image test set was composed from color and grayscale images ranging from face to aerial images. The size of the images was 1024 × 768 pixels. For testing we have used data payload of 2500 bits, therefore, relative payload size was $p = 0.003$ bpp (bits per pixel). First, we empirically determined the optimal DCT coefficient in a block for the implementation. Then, we found what implementation factor should be used, and, finally, the optimized proposed method was tested for robustness against different image processing attacks.

4.1 Influence of the position of the modified coefficient

The performance of the method in terms of zhe robustness and the impact on the quality of the stego image depends on the implementation factor α and the position of modified DCT coefficient. To determine which coefficient should be modified for optimal results, we have embedded the data payload in each of the 64 DCT coefficients and compared cover and stego images using the Structural Similarity (SSIM) index proposed in [28] and assessed in [11]. The mean value of the SSIM index for corresponding coefficients of all images was calculated. As can be seen in Fig. 6, the impact on the image quality correlates well with the frequency of a DCT coefficients. Very high degradation of the image happens when the DC component of the image is modified (SSIM ≤ 0.5). In addition, this degradation occurs even for low values of the implementation factor α. For other coefficients, the quality of the stego image improves as the frequency is increased. However, it is interesting to note that SSIM values slightly drop for highest horizontal and vertical frequencies (Fig. 6).

For the influence of the position of the modified DCT coefficient on the robustness of the method we use the Bit Error Ratio (BER). Again, we have embedded the data payload in each of the 64 DCT coefficients, after which the stego image with the hidden data payload was degraded by different attacks, namely, blur, unsharp masking, JPEG compression with different quality levels, and scaling. After each attack BER was calculated by comparison of the original data payload, and the extracted data payload. Figure 7 shows mean BER values of image test set for all attacks.

4.2 Influence of the implementation factor

From Fig. 6 and Fig. 7 it is obvious that the exact coordinates of a modified coefficient depends on the imple-

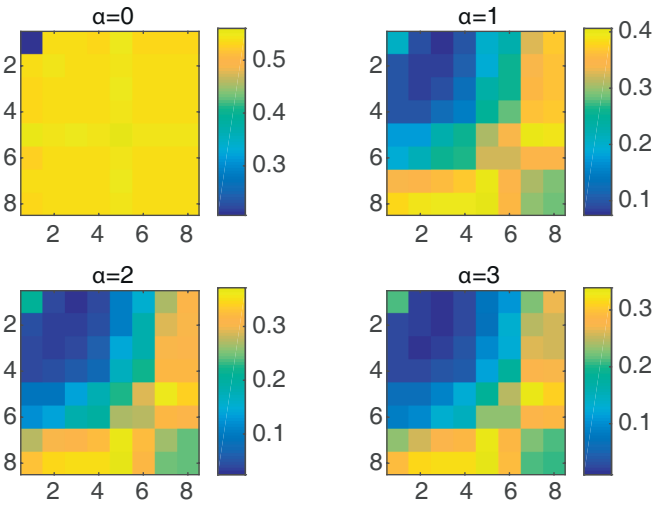

Fig. 7 Impact of the position of a modified DCT coefficient on the robustness of the method against attacks for a given implementation factor α. BER increases as the frequency of the modified DCT coefficient gets higher.

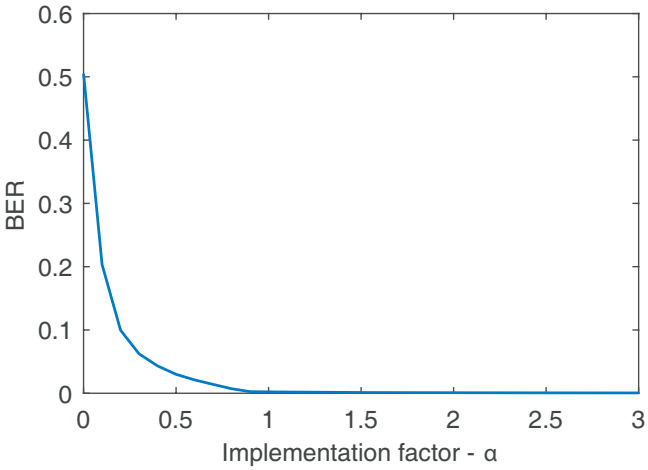

Fig. 8 Influence of the implementation factor on the decoder performance for images before attack.

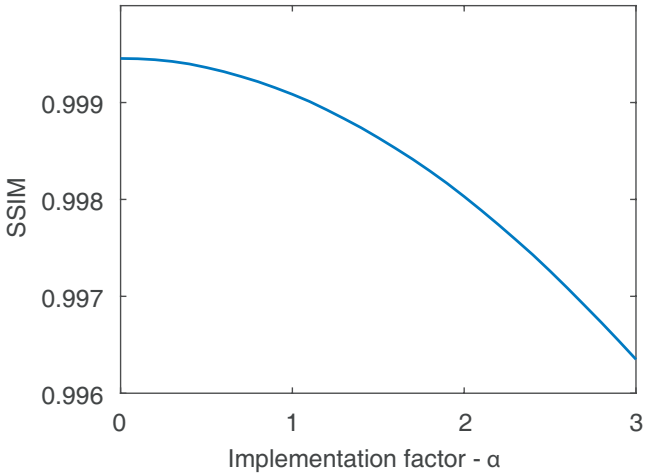

Fig. 9 Influence of the implementation factor on the quality of the stego image.

mentation factor α. To avoid visible degradation of the stego image quality, for the higher values of α one should modify only high frequency coefficients. However, high frequency coefficients are very sensitive to image processing operations such as JPEG compression and bluring. Therefore, to further the investigation and determine the optimal implementation factor we set the coordinates of the DCT coefficient to be modified to $M(4, 4)$.

To determine the optimal implementation factor for the aforementioned DCT coefficient we conduct two experiments. In order to determine the baseline robustness of the method, we embed hidden message using a range of implementation factor values, and calculate BER on stego images before image processing attacks. Second, for same images, we calculate SSIM to determine the degradation of the stego image due to information hiding.

The first test was BER test on un-attacked images. It was used to determine how high the factor of implementation must be in order to retrieve the hidden information back from the stego image without any loss. The results showed that the factor of implementation must be at least 1 to fully recover embedded information (Fig. 8).

The test of the degradation of the stego image was conducted by calculating the influence of the implementation factor on the SSIM value. As depicted in Fig. 9 the mean SSIM value for image test set falls below 0.998 for implementation factor above 2. It is interesting to note that the graph very closely follows negative quadratic function.

4.3 Robustness of the method

To determine how robust is the proposed method, different attacks were conducted and the attacked images were tested. Blur 3x3, Unsharp mask, JPEG 25, JPEG 50, JPEG 75, Scale 0.5 and Scale 2 were the attacks that were used on a database of 140 images. The attacks represent different categories of attack combining both, unintentional and malicious attacks. This is only a rough classification and most attacks, depending on the intention, can fall into both categories. It should be noted that the proposed method is not robust against synchronization attacks such as cropping, reflection, shearing or rotation.

The results of the testing of the robustness are shown in the table 1 where mean, median, minimum, maximum and standard deviation BER valus are given. JPEG 25 being the most aggressive attack gave the worst results with the mean and median BER value of 0.32. Large

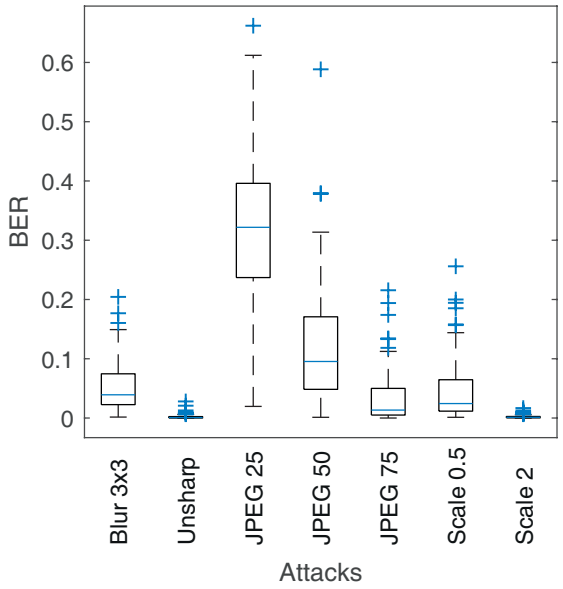

Fig. 10 Boxplot of BER values after attacks.

Table 1 BER values after attacks.

Attack	BER				
	mean	median	min	max	std
Blur 3×3	0.05	0.04	0.00	0.20	0.04
Unsharp	0.00	0.00	0.00	0.03	0.00
JPEG 25	0.32	0.32	0.02	0.66	0.13
JPEG 50	0.12	0.09	0.00	0.56	0.10
JPEG 75	0.03	0.01	0.00	0.21	0.04
Scale 0.5	0.04	0.02	0.00	0.26	0.05
Scale 2	0.00	0.00	0.00	0.02	0.00

standard deviation is due to a wide range of different images tested. For some images, the BER value is low (minimum BER = 0.02) regardless of a very aggressive degradation. The lowest BER value of the decoded hidden data payload was obtained for unsharp and enlargement (Scale 2) attacks where the mean BER value is 0. The boxplot of BER values for all attacks (Fig. 10) clearly shows that the proposed method is very robust to moderate attacks (JPEG 50, Scaling, Blur), and somewhat robust to aggressive attacks such as JPEG 25.

4.4 Steganalysis of the method

For the steganalysis of the method we used the state of the art method proposed in [17]. This method is an ensamble classifier built with a large number of base learners. Final decision of the classifier is made by colecting votes of all base learners. Base learners use a randomly selected subset of the feature set. To bild a feature set for the classifier, we followed approach proposed in [22], and build the feature set by merging extracted calibrated Markov features [26] and extended DCT features.

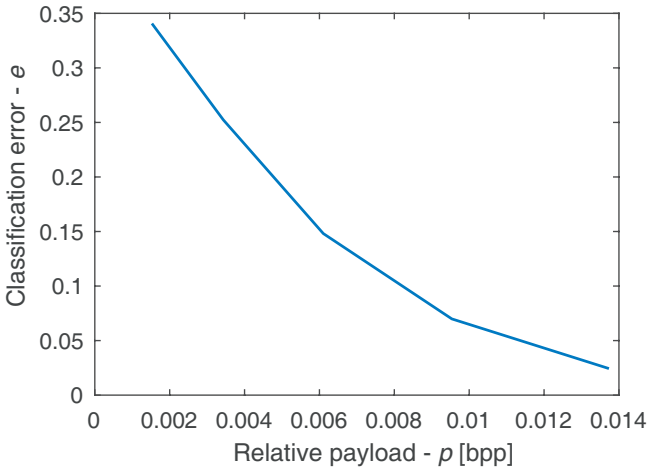

Fig. 11 Classification error of the classifier for different relative payload sizes.

For the training of the classifier we embedded the hidden payload of different relative sizes in 1000 images (implementation factor was set to $\alpha = 1$, $M(4,4)$ DCT coefficient was modified). The classifier learns by comparing the extracted features of the cover images with the extracted features of the stego images. The testing was done on 1000 images previously unseen by the classifier. As expected, the performance of the classifier depends on the relative payload size p (Fig. 11). For the relative payload size $p = 0.003$ bpp (the amount of data embedded in previous experiments) the classification error is around $e \approx 0.25$. For larger payloads the classification error decreases. This means that the proposed steganographic method is valid for smaller relative payload sizes p.

The validity of the proposed method for smaller relative payload sizes is obvious from the ROC performance curve of the classifier against our method (Fig. 12). When the relative payload size is sufficiently small ($p < 0.003$ bpp), the false positive probability of the classification become expensively high, i.e. there will be a lot of false alarms at the classifier. This advantage can be improved by a further decrease of the relative payload size.

5 Parallel Implementation

As mentioned previously, OpenCL is used to accelerate the program written in C to get better performance and faster execution. The parallelization process is carried out at the kernel level which corresponds with the main algorithm steps: RGB2YCbCr, DCT, the embedding process, the extraction process, and the inverse modules. Optimizations are based mainly on the efficient exploitation of both data parallelism and memory hierarchy.

Specifically, in the RGB2YCbCr and YCbCr2RGB modules, the data level parallelism is exploited by par-

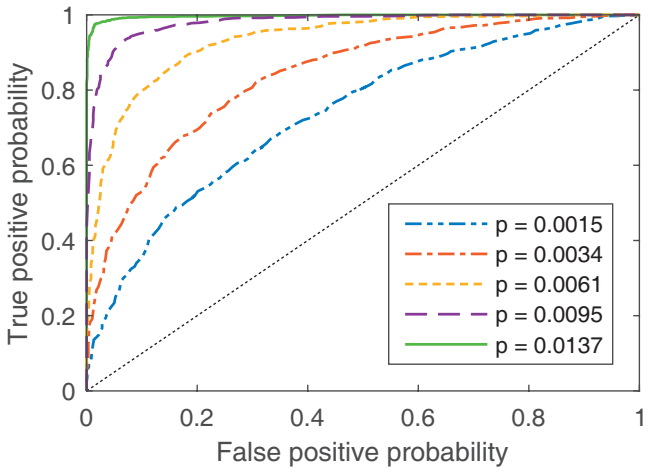

Fig. 12 ROC performance curves of the classifier for different relative payload sizes.

titioning the original image at pixel level, and grouping pixels into *work-groups*. Due to non-existent data reuse in the process, it is not worth to take advantage of the OpenCL hierarchy memory. The only optimization regarding the memory is the data memory alignment by means of the array padding technique.

DCT and iDCT OpenCL implementation are based on CUDA's DCT8x8 example available in CUDA Image Samples [9] but fully developed in OpenCL. Input image is usually divided into macroblocks of 8×8 pixels. DCT8x8 transform is usually optimized by avoiding redundant multiplications and utilizing the separability of 2D transform. DCT8x8 computation reduction is based on the approach proposed in [27] in which the greatest improvement is due to the floating point division performed as constant reciprocal multiplications. Memory exploitation is also performed by the OpenCL memory hierarchy usage in combination with bank conflict reduction at local memory. Most memory access is performed in coalesced manner (load-store pattern operations are done in consecutive memory addresses) by means of local memory.

Extract and embed stages are translated into kernels in the easiest way. The implementation is based on the distribution of embed image values between all *work-items* available. Motivated by the simple operations involved in embed/extract of a single data payload value, more aggressive optimizations are not required.

6 Target Architectures

In this section, we present the main features of the architectures used in the evaluation of performance acceleration of the selected encoder and encoder+decoder implementations. We have organized our implementation on two desktop configurations (Nvidia GTX980 and AMD-APU) and one low-cost and low-power architecture (Odroid-Mali). A general-purpose processor (Intel Xeon) is also used, and it will be evaluated as a baseline for our performance measurements and conclusions.

Table 2 gives an overview of the main features of the target architectures. In the rest of the study, we will refer to each architecture using the name reported in the table.

6.1 Hardware accelerators

Today, the usage of massively parallel hardware accelerators is playing a key role in the HPC arena. Many of the most powerful supercomputers are today equipped with one or more accelerators per computing node. In some cases, even low-end desktop computers exhibit relatively powerful graphics hardware, that can be exploited to accelerate critical applications.

Nvidia Maxwell: The Nvidia GeForce GTX 980 is a graphic card with 4 Gb of memory with a core speed of 1126 Mhz and memory speed of 1750Mhz (7000Mhz effective). It has a 256-bit bus width and consumes 165 W. These platforms are able to easily exploit the inherent data parallelism available in some applications with a relatively small programming effort.

AMD-APU: AMD A10-6800K APU [2] has a 4 core proccesor on 4.1 Ghz with a 4.4 Ghz boost option, it has 4MB of total L2 cache and consumes 100 W. AMD HD6870D [3] is a integrated graphics chip on AMD A10-6800K APU, it has core speed of 844 Mhz and a memory clock at 1067 Mhz, 128-bit bus width and a power consumption of 100W.

6.2 Low-power architectures

The increasing processing demands of the mobile market, together with the necessity of designing highly-efficient architectures, have contributed to the development of processing platforms in which *Performance per Watt* is the primary design concern. This type of systems, often based on the ARM architecture with some accelerating platform attached (low power GPUs or DSPs) has attracted the attention of the HPC community, and has emerged as an appealing alternative for scenarios in which the high performance is needed, but the power efficiency or the maximum available power is a critical restriction. We report next the low-power architectures employed in our study.

Odroid-Mali: The ARM Cortex is a line of RISC processors that implements a dual-issue superscalar, out-of-order pipeline. In our case, we use an Odroid XU3 board, featuring a Samsung Exynos 5422 heterogeneous SoC that implements the ARM big.LITTLE hybrid architecture: it presents two independent processing clusters, the

Table 2 Summary of the main features of the target architectures regarding OpenCL programming paradigm.

	Intel Xeon CPU E3-1225	NVIDIA GeForce GTX 980	AMD A10-6800K APU	MALI Odroid
OpenCl version	1.2	1.2	1.2	1.1
Max. Compute Units	4	16	4	4
Local Memory Size	32 KB	47 KB	32 KB	32 KB
Global Memory Size	32101 MB	4095 MB	7197 MB	2048 MB
Max WorkGroup Size	8192	1024	1024	256
Max Work-item Dims	(8192 8192 8192)	(1024 1024 64)	(1024 1024 1024)	(256 256 256)

first with four ARM Cortex A7 ultra-low power cores and the second with four ARM Cortex A15 cores. Each cluster owns an independent cache hierarchy, while the main DDR memory is shared between the both. The selection of the specific processing cluster can be modified at runtime depending on the experimental necessities. It also includes a Mali-T628 MP6 (OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile). These type of processors are the base of many current mobile devices (including phones, tablets, hand-held devices and even low-power desktop systems), hence the interest in studying its feasibility for signal and image processing applications like those using steganography, in which both the mobility and power consumption can be a concern.

6.3 General-purpose processors

Intel Xeon E3: We will also evaluate the efficiency of our implementations both in terms of performance and power consumption for a general-purpose Intel Xeon processor, using it as a baseline. More specifically, we use an Intel Xeon E3-1225 v3 featuring with a quad-core socket and running at 3.2 Ghz, with 16 Gbytes of DDR3 RAM. We consider this processor to be the representative of common modern multi-core architectures for desktop and server environments.

7 Results

Steganographic methods consist of a flow of functions that need to be executed in a specific order. However, in that flow, each of the functions can be implemented in a different way in order to take advantage of the specific characteristics that each target architecture can offer. Due to this, it is interesting to know the way the execution time during the encoder and decoder operations of the proposed steganographic method is distributed among the different functions. This information will allow to determine which functions can benefit more from the existence of high performance mechanisms in different architectures.

Using as baseline the Intel Xeon E3-1225 processor without any acceleration mechanism, single-thread mode, the execution time spent in each of the functions in the encoder and decoder operations has been measured. Fig. 13 and Fig. 14 summarize the results for the baseline architecture using a collection of images of increasing size. The time is expressed as the percentage of time that each functional block is being executed. Image size corresponds to several digital images resolution (720×400 for DVD, 1280×720 for HD-DVD, 1920×1080 for Blueray, 2048×1080 for 2K-Digital Cinema, 3840×2160 for 4K Ultra-high-definition television, 4096×2160 for 4K-Digital Cinema and 7680×4320 for 8K Ultra High Definition).

According to Fig. 13, all the functions involved in the encoder operation behave similarly as the image size increases, keeping analogous relative percentages. The figure shows that the most time consuming function is transforming RGB into YCbCr color space, requiring between 50% and 60% of the total execution time. Around 30% of the time is spent on the reverse process (YCbCr to RGB). The remaining functions represent, in total, only between 10% and 15% of the total execution time. The transformations in/from frequency domain consume a similar amount of time, while the process of embedding data in the images is almost negligible.

Similarly to the encoder operation, the functions involved in the decoder process show the same trend in the baseline architecture. The most time consuming step among the existing functions is the transformation of RGB into YCbCr color space, around 80% of the time. The negligible step in the decoder process in the vector extraction.

Fig. 13 and Fig. 14 indicate that although high performance techniques can be applied to any of the functions involved in the steganographic methods, the most significant improvements will be achieved reducing the execution time during the color space conversions.

Regarding accelerators such as Xeon E3 and Geforce 980 it can be seen that stages for data transformation from RGB into YCbCr color space is again the most time consuming. This results are depicted in Fig. 15 and Fig. 16 for the Xeon E3 and Fig. 17 and Fig. 18 for the Geforce GTX 980.

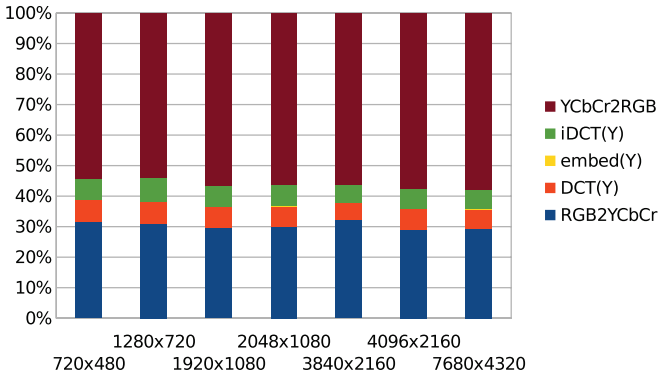

Fig. 13 Encoder Profiling trough different stages. Baseline.

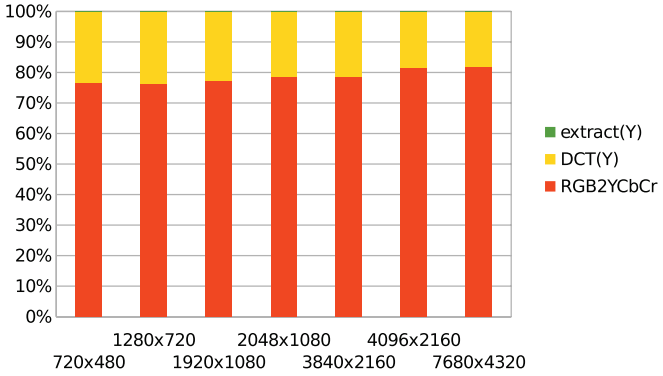

Fig. 14 Decoder Profiling trough different stages. Baseline.

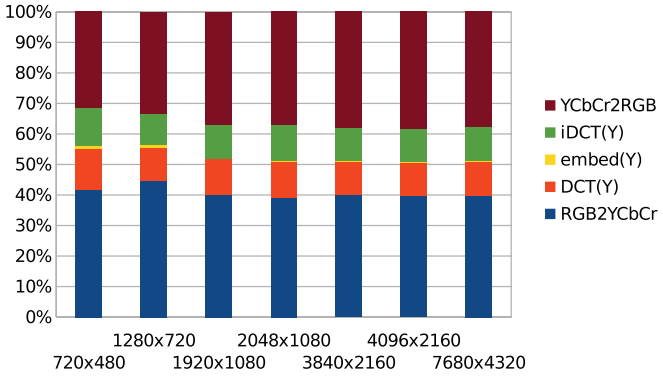

Fig. 15 Encoder Profiling trough different stages. Xeon E3.

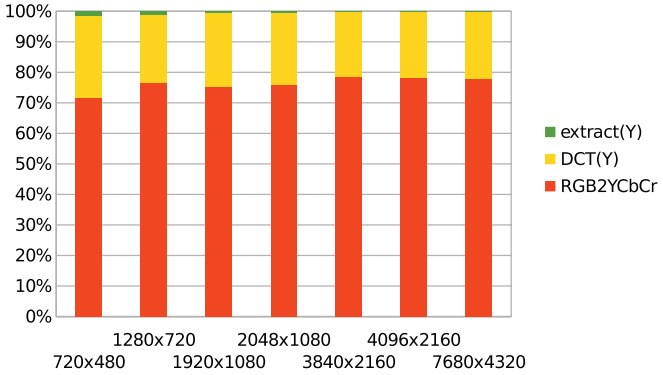

Fig. 16 Decoder Profiling trough different stages. Xeon E3.

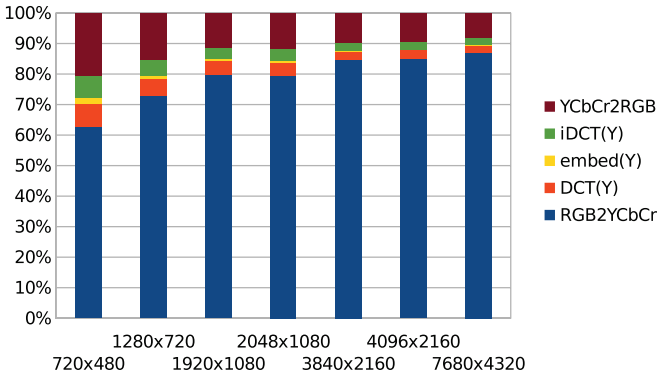

Fig. 17 Encoder Profiling trough different stages. Geforce GTX 980.

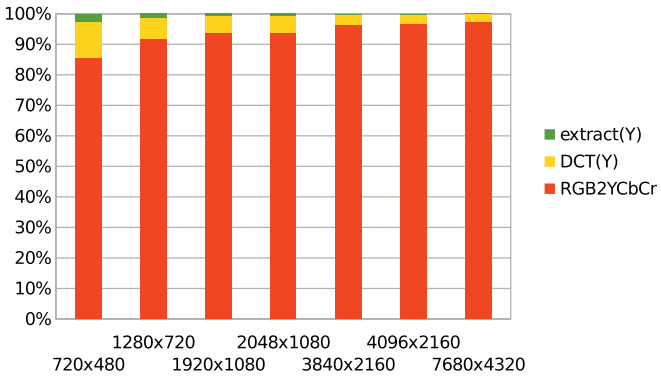

Fig. 18 Decoder Profiling trough different stages. Geforce GTX 980.

7.1 High performance results

Having seen the distribution of the execution time on the baseline architecture during the encoding and decoding processes, it is time to apply the different high performance mechanisms existing in the target architectures in order to know their impact on time. The baseline architecture that will be used for comparison purposes is the same as in previous subsection, the Intel Xeon E3-1225 processor without any improvement mechanism. The architectures to compare with will be the hardware accelerators Nvidia GeForce GTX 980 and AMD A10-6800K APU and the low-power architecture ARM Odroid-Mali and AMD Radeon. The Intel Xeon E3-1225 processor, taking advantage of their high performance mechanisms, is also included in the comparison.

The results of the absolute execution time of the encoder and decoder are shown in Fig. 19 and Fig. 20 respectively. The speedups obtained for each architecture compared to the baseline architecture are shown in Table 3 for the encoding process, while Table 4 shows the speedups for the decoding process.

Regarding the encoder, Fig. 19 shows in each architecture, even in the baseline, a kind of logarithmic behaviour in execution time as the image size increases. Low-power architectures show the worst execution time

Table 3 Encoder speedups.

	720x480	1280x720	1920x1080	2048x1080	3840x2160	4096x2160	7680x4320
IntelXeonE3	3.16	3.66	3.89	3.85	3.72	3.69	3.51
GeForce GTX980	5.27	7.66	9.51	9.86	13.65	12.07	13.07
ARM-Mali	0.69	0.75	0.74	0.74	0.85	0.74	NA
AMD-Radeon	0.14	0.32	0.59	0.65	1.82	1.65	1.44

Table 4 Decoder speedups.

	720x480	1280x720	1920x1080	2048x1080	3840x2160	4096x2160	7680x4320
IntelXeonE3	2.45	2.41	2.28	2.28	2.22	2.62	2.48
GeForce GTX980	2.87	3.32	3.56	3.69	4.23	4.99	5.27
ARM-Mali	0.41	0.43	0.41	0.41	0.41	0.47	NA
AMD-Radeon	1.56	1.54	1.54	1.57	1.62	1.99	1.24

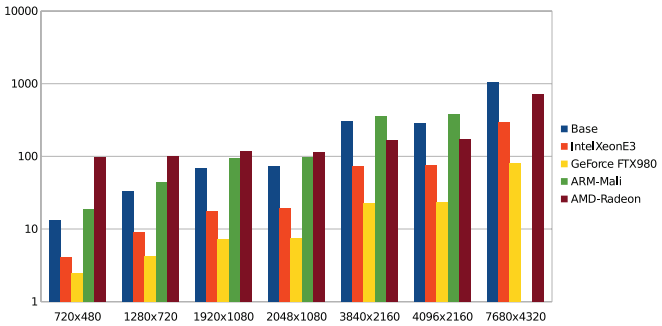

Fig. 19 Encoder time (miliseconds) trough different platforms.

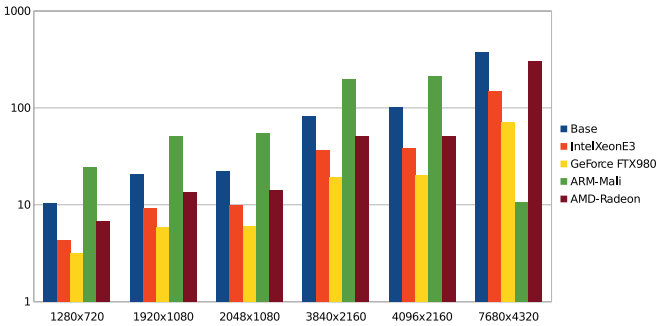

Fig. 20 Decoder time (miliseconds) trough different platforms.

among the target architectures. Their power restrictions limit the performance of the accelerating mechanisms included in the platforms. Despite this behaviour, the ARM Mali can still be an option if power consumption matters, since the final execution time is just a bit longer than the baseline for all the image sizes. However, AMD Radeon behaves worse for small image sizes, only improving the baseline time for large image sizes. The Intel Xeon processor shows a constant speed up for all the image sizes in comparison to its baseline version. Nvidia GeForce GTX 980 performs the best execution time, with increasing speedups as the image size increases.

The execution time during decoder operations shows similar trends to the encoding time. Constant speedups are detected for the Intel Xeon processor, which reduce the execution time in half. In this case, ARM Mali behaves worse than for the encoder, almost doubling the execution time of the operation compared to the baseline. However, AMD Radeon behaves better when decoding, since the improvement in time happens for all the image sizes. Finally, the Nvidia GeForce GTX 980 accelerator is the architecture which performs the best behavour, although achieving lower speedups for decoder than for the encoder.

8 Conclusion

The proposed steganographic method has a large capacity, and offers strong robustness for moderate attacks. In addition, it has a low impact on the image quality. Security of the method enhanced by random dispersion of the hidden data and key for the pseudo-random number generator. The modification of medium frequency coefficients offers good balance between robustness and perceptibility. In addition, factor α can be used to optimize the method for specific application. The steganalysis of the method showed that the relative data payload size should be kept below $p = 0.003$ bpp.

Analysis of the implementation of the proposed method on different architectures shows that the method can be used in real-time applications on more powerful accelerators like Nvidia GeForce GTX 980 achieving 4K real-time video encoding (43 fps), as well as on low-power architectures as Odroid XU3 targeting HD-DVD video coding (25 fps).

Future research will be focused on further optimization of the method depending on specialized applications such as hiding of biometric data in ID images. Also, FPGA implementation of the method and analysis of the performance on FPGA platform will be conducted.

References

1. Alvarez P (2004) Using Extended File Information (EXIF) File Headers in Digital Evidence Analysis International Journal of Digital Evidence. International Journal of Digital Evidence 2(3):1–5
2. AMD (2015) AMD A series APU processors,15.6.2015. http://www.amd.com/en-us/products/processors/desktop/a-series-apu
3. AMD (2015) AMD Radeon HD 8670d, 15.6.2015. http://www.pc-specs.com/gpu/AMD/APU_Family/Radeon_HD_8670D/1787
4. Bamatraf A, Ibrahim R, Salleh MNBM (2010) Digital watermarking algorithm using LSB. In: 2010 International Conference on Computer Applications and Industrial Electronics, IEEE, Iccaie, pp 155–159, DOI 10.1109/ICCAIE.2010.5735066, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5735066
5. Bhatnagar G, Raman B (2009) A new robust reference watermarking scheme based on DWT-SVD. Computer Standards & Interfaces 31(5):1002–1013, DOI 10.1016/j.csi.2008.09.031, URL http://linkinghub.elsevier.com/retrieve/pii/S0920548908001499
6. Celik M, Sharma G, Tekalp A, Saber E (2005) Lossless generalized-LSB data embedding. IEEE Transactions on Image Processing 14(2):253–266, DOI 10.1109/TIP.2004.840686, URL http://www.ncbi.nlm.nih.gov/pubmed/15700530http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1381493
7. Cheddad A, Condell J, Curran K, Mc Kevitt P (2010) Digital image steganography: Survey and analysis of current methods. Signal Processing 90(3):727–752, DOI 10.1016/j.sigpro.2009.08.010, URL http://dx.doi.org/10.1016/j.sigpro.2009.08.010http://linkinghub.elsevier.com/retrieve/pii/S0165168409003648
8. Cox I, Miller M, Bloom J, Fridrich J, Kalker T (2008) Digital watermarking and steganography, 2nd edn. Morgan Kaufmann Publishers, San Francisco, CA, USA
9. CUDA (2016) CUDA Toolkit Documentation - sample reference. http://docs.nvidia.com/cuda/cuda-samples
10. Dey S, Abraham A, Sanyal S (2007) An LSB Data Hiding Technique Using Prime Numbers. In: Third International Symposium on Information Assurance and Security, IEEE, 1, pp 101–108, DOI 10.1109/IAS.2007.37, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4299758
11. Dosselmann R, Yang XD (2011) A comprehensive assessment of the structural similarity index. Signal, Image and Video Processing 5(1):81–91, DOI 10.1007/s11760-009-0144-1, URL http://link.springer.com/10.1007/s11760-009-0144-1
12. Groups K (2016) OpenCL Consortium. URL https://www.khronos.org/opencl
13. Hashad A, Madani A, Wahdan AMa (2005) A Robust Steganography Technique Using Discrete Cosine Transform Insertion. In: 2005 International Conference on Information and Communication Technology, IEEE, Cairo, pp 255–264, DOI 10.1109/ITICT.2005.1609628, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1609628
14. Kang X, Huang J, Member S, Zeng W (2010) Efficient General Print-Scanning Resilient Data Hiding Based on Uniform Log-Polar Mapping. IEEE Transactions on Information Forensics and Security 5(1):1–12, DOI 10.1109/TIFS.2009.2039604, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5378607
15. Kedmenec L, Poljicak A, Mandic L (2014) Copyright protection of images on a social network. In: Proceedings ELMAR-2014, IEEE, September, pp 1–4, DOI 10.1109/ELMAR.2014.6923345, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6923345
16. Kee E, Johnson MK, Farid H (2011) Digital Image Authentication From JPEG Headers. IEEE Transactions on Information Forensics and Security 6(3):1066–1075, DOI 10.1109/TIFS.2011.2128309, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5732683
17. Kodovský J, Fridrich J, Holub V (2012) Ensemble classifiers for steganalysis of digital media. IEEE Transactions on Information Forensics and Security 7(2):432–444, DOI 10.1109/TIFS.2011.2175919
18. Kurak C, McHugh J (1992) A cautionary note on image downgrading. In: [1992] Proceedings Eighth Annual Computer Security Application Conference, IEEE Comput. Soc. Press, pp 153–159, DOI 10.1109/CSAC.1992.228224, URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=228224http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=228224
19. Lu W, Sun W, Lu H (2009) Robust watermarking based on DWT and nonnegative matrix factorization. Computers & Electrical Engineering 35(1):183–188, DOI 10.1016/j.compeleceng.2008.09.004, URL http://linkinghub.elsevier.com/retrieve/pii/S0045790608001067
20. Naveed I, Puech W (2013) Data Cryptography. In: Signal and Image Processing for Biometrics, John Wiley & Sons, Inc., Hoboken, NJ, USA, pp 263–277, DOI 10.1002/9781118561911.ch13, URL http://doi.wiley.com/10.1002/9781118561911.ch13
21. Parthasarathy A, Kak S (2007) An Improved Method of Content Based Image Watermarking. IEEE Transactions on Broadcasting 53(2):468–479, DOI 10.1109/TBC.2007.894947, URL http://ieeexplore.ieee.org/lpdocs/epic03/

wrapper.htm?arnumber=4215117
22. Pevny T, Fridrich J (2007) Merging Markov and DCT features for multi-class JPEG steganalysis. Proceedings of SPIE 6505:650,503, DOI 10.1117/12.696774, URL http://link.aip.org/link/?PSISDG/6505/650503/1
23. Poljicak A, Mandic L, Agic D (2011) Discrete Fourier transformbased watermarking method with an optimal implementation radius. Journal of Electronic Imaging 20(3):033,008, DOI 10.1117/1.3609010, URL http://link.aip.org/link/JEIME5/v20/i3/p033008/s1{&}Agg=doi
24. Scarpino M (2012) OpenCL in Action: How to Accelerate Graphics and Computation. ny
25. Schneier B (1995) Applied cryptography: Protocols, algorithm, and source code in C, 2nd edn. John Wiley & Sons
26. Shi Y, Chen C, Chen W (2007) A Markov process based approach to effective attacking JPEG steganography. Information Hiding pp 249–264, DOI 10.1007/978-3-540-74124-4, URL http://www.springerlink.com/index/c308w1674110v420.pdf
27. Sung T, Shieh Y, Yu C, Hsin H (2006) High-efficiency and low-power architectures for 2-d dct and idct based on cordic rotation. In: Parallel and Distributed Computing, Applications and Technologies, 2006. PDCAT '06. Seventh International Conference on, pp 191–196, DOI 10.1109/PDCAT.2006.70
28. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP (2004) Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing 13(4):600–612, DOI 10.1109/TIP.2003.819861