HPCLatAm 2013, pp. 122-133 (full paper)
Session: GPU Architecture and Applications
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Encrypting video streams using OpenCL code on-demand

Juan P. D'Amato[1,2], Marcelo Vénere[1]

[1]PLADEMA, Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires .
[2]Consejo Nacional de Investigaciones Científicas y Técnicas, Rivadavia 1917, Ciudad Autónoma de Buenos Aires, Argentina

Abstract. The amount of multimedia information transmitted through the web is very high and increasing. Generally, this kind data is not correctly protected, since users do not appreciate the information that images and videos may contain. In this work, we present an architecture for managing safely multimedia transmission channels. The idea is to encrypt and encode images or videos in an efficient and dynamic way. The main novelty is the use of on-demand parallel code written in OpenCL. The algorithms and data structure are known only at communication time what we suppose increases the robustness against possible attacks. We conducted a complete description of the proposal and several performance tests with different known algorithms.

Keywords: Parallelism & GPU, Encryption, on demand

1 Introduction

Recent advances in technology and communications have led to the uncontrolled raising of multimedia data consumption through the WEB. These advances also are companied by a radically change in the way of delivering and accessing data. Today, there are many applications in many areas such as High-definition TV, home automation, video-conferencing, among many others, which are accessed using different devices, from smart-phones to high-performance PCs. This context has favored the unsafe access, fraud, attack and robbery of this data.

Unfortunately, in transmission and multimedia processing, the amount of data to be transmitted is prioritized against security; no matter that these data are highly confidential and personal. It can be argued that current known platforms as OpenSSL or SRTP cover these aspects, using known encryption schemes. But these schemes are inefficient for image/video transmission; they have been designed for generic uses and do not always consider the different user processing capabilities. They also have not considered the new ways of storing data such as *clouds* [14] where the data host, which is not the data owner, can eventually expropriate and manipulate data. A criti-

cal case is imaging repositories usually called PACS, which usually have many users visualizing information [11].

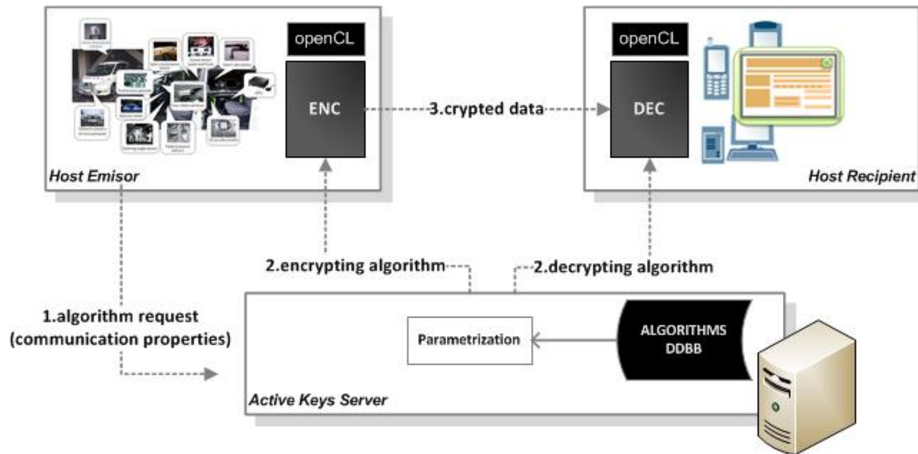

Fig. 1. Proposed Architecture

These performance, distribution and adaptability to different devices issues have motivated our proposal. The idea is to develop a platform that cans adaptive select algorithms and encryption keys for each new connection in accordance with processing capabilities. At the same time, it is ensured that the host does not know beforehand how data is encoded. To let these algorithms run efficiently, they are coded in OpenCL which is a standard for high performance computing. These algorithms, along with the encryption keys are distributed as a script key, concept that we have called *active keys*.

In order to increase data reliability, it is proposed to separate the host that stores the keys from the encryption mechanism; so a third participant is included, an *active key server* responsible for managing such keys. New and more robust encryption algorithms, as long as, visualizing and encoding methods can be included, generating a enriched strategies database. Throughout the paper, we present the proposed architecture and some performance results obtained from image processing. A security analysis was not necessary, as we propose to use known and validated algorithms.

2 Background

There are many studies and reviews related to multimedia data encryption. Some works such as [15] suggest encrypting data using classic algorithms such as AES or DES; scheme known as 'naive algorithm'. SRTP, presented in [5] as well as some versions of DTV, apply this kind of solution. These families of strategies are very

inefficient, because the algorithms used are generic and they have been designed for small blocks of data.

When large volumes of data transmissions must be processed, such as images and video, some authors propose to partition the data and process them in parallel [10]. In some cases, graphics cards (GPUs) are used to obtain encryption in real time (about 30 frames per second). Other solutions propose extending classical video coding algorithms, such as MPEG, adding an encryption step [6,13]. These solutions that combine encoding with encryption are called Joint Video Compression Encoding (JVCE). Many recent works are variations on this idea [9].

As it's called in [7], these proposals have some limitations and there is not one scheme that can meet all specific requirements. Some solutions are tied to specific hardware platforms and they cannot run on any device. In other cases, the algorithms used are pre-defined and do not take into account the characteristics of the communication, such as image size, processing speed, among others, causing inefficient use of resources.

The last issue and perhaps the one we consider the most serious is that the encryption technique is generally known. This condition allows data to be decoded if attacked, even using brute force [8]. To resolve similar issues, in the work of [12] is introduced the idea of 'encryption on demand', who proposes that the encryption is done using encryption keys within a JavaScript (JS). This script is downloaded from a server and it is known only during communication time. This scheme increases the security of the data, but the process is very slow just because JS is interpreted.

In this work, we propose to use OpenCL API for running the encryption algorithms in parallel, both on CPUs or GPUs. Below, some features of OpenCL that were taken into account in this proposal are discussed.

3 OpenCL API

The advent of multi-core platforms and massive parallelism with GPUs bring the advantage (and disadvantage if they're used in a malicious manner) to increase computing capacity per unit time. This increased capacity calculation can be exploited especially in applications where data can be partitioned and processed in parallel. Indeed, several studies have proposed encrypt or encode data using GPUs, achieving good performance in real time even for high quality video [4].

On the other hand, the proposed solutions often use CUDA as a development platform, limiting its function to PCs (desktops or clusters). Another alternative to achieve parallelism on different platforms is OpenMP that runs on Linux, Windows, iOS and Android, but it has the limitation of using only pre-compiled code.

The other parallelization technology is OpenCL proposed by Khronos Group, that has quickly gained popularity and it is being also adopted in Web browsers. This standard uses a programming language ANSI C, which is loaded dynamically and compiled into one device. Each program is instantiated in a method or *kernel*, which runs in multiple threads within a computing unit. Each kernel accesses memory spaces in the three levels, as shown in **Figure 2**.

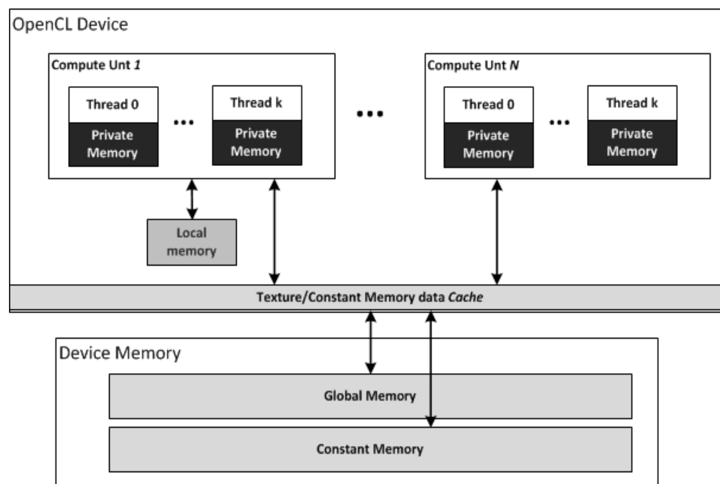

Fig. 2. OpenCL memory scheme

Although it has not all the facilities that have other platforms such as CUDA, the biggest advantage of OpenCL is portability, either GPU or CPU. This advantage let us distribute efficient and mobile code, where algorithms can be transmitted and compiled on devices with different features. Surely, as it's a standard under development there are certain capabilities not supported by all architectures, but we understand it is a limitation that will be overcome with time.

4 Methodology

As in any secure communication scheme, there exists the transmitter, the channel and the receiver of the communication. For a new data request, multimedia in this case, both transmitter and receiver should define the data structure and the encryption mechanism.

In this paper, we extend this basic configuration including a third participant called the *active key provider* In this architecture, each new communication between the parties also involves an algorithm encryption/decryption request to the server, which is responsible for selecting a suitable algorithm and corresponding keys for the encryption. These algorithms are coded as a script in ANSI C for OpenCL and they should be safely transmitted to the participants.

On the client side, after receiving the code, the platform will prepare to process with an OpenCL module. The necessary kernels are created and compiled, memory spaces are allocated if required and finally the kernels are executed. The incoming data, e.g. a frame from a video, are loaded and processed in the module before sending through the net. In the same way, the receiver loads and compiles decoding

scripts, and processes the data stream as it comes. When communication finishes, the channels are closed and the active keys are removed.

The encryption steps are shown in Figure 3:

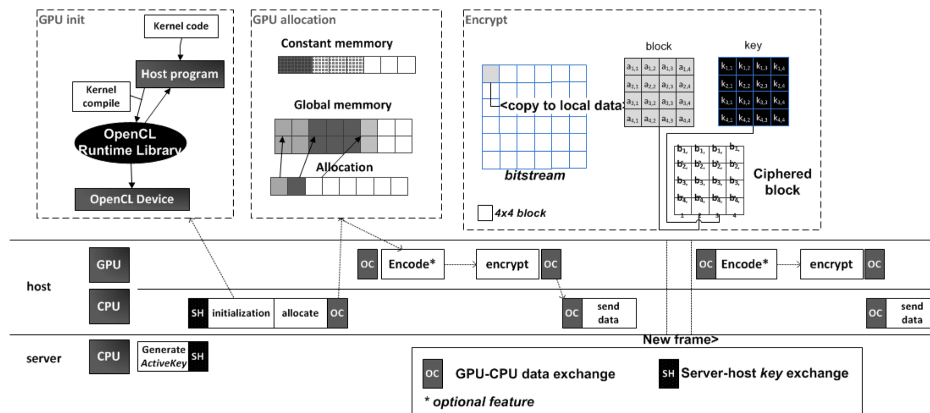

Fig. 3. : Server-host encryption scheme

So far, we have given the general idea of this proposal. Now, we explain how the server should manage the algorithms, and then we describe their structure. Although we have focused on encryption, the same schema can also be applied to compress videos and pictures.

4.1 Active Keys Management

The key management in a cryptography environment involves the generation, exchange, store and use of passwords. This task is critical for safety and it is one of the most complex issues to address. In the case of active keys, where the encryption mechanism is now conformed by the key plus the algorithm, these tasks should take into account particular features of the architecture and the data domain.

Regarding key generation, it is important to generate them in a safe and pseudo-random way. For example, a key with many 0s is unacceptable and good generating strategies are required. In our architecture, keys can be generated on both the client and server side. This step will be explained in the next section.

Respect to the keys exchange, some of the known schemes as Diffie-Hellman Key Exchange protocol, Key Wrap or RSA [3] can be applied. In this paper we do not propose any particular one, since it can be adapted to the architecture. Regarding storage and use of keys, our proposal is further differentiated from existing ones. The database containing the algorithms can grow as new algorithms are included, as the one proposed in [1]. Thanks to this, at the time of a request, the server can choose among plenty of algorithms, reducing vulnerability to attacks. The new techniques must be implemented with a certain code template and they should also include a description of auxiliary structures; both are described in the next section.

It's important to remark that although OpenCL limits access to resources such as hard disks or peripherals, it does not limit memory access, making it vulnerable to out of range exploits. To ensure that the algorithms incorporated to the database are valid, they must be subjected to a series of tests with different data types and sizes.

4.2 Keys and structures generation

One of the most important steps before encryption is the algorithms parametrization. In general, the encryption keys used, either symmetric or asymmetric, are long numbers having 128, 256 or more bits. Other algorithms, such as AES, also require structures such as Rijndael dictionaries [2], conversion functions, among others. Faced with a new communication, these structures can be generated either server side or client side.

If the data are generated on server-side, it is proposed that these be must be included in the algorithm script as a constant structure with values from a base type (char or integer). To let the server automatically include the structures within the *script*, we propose to use "tags" in the code. These tags indicate which parameters should be generated and each one should correspond to a generation method. It can also be added a random *seed*, used for inner methods. In **Table 1**, we suggest some parameters and their *tags* included in the platform.

Table 1. : Algorithms parameters with their corresponding *tags*

Cipher	Parameters	Size (Bits)	Tag
AES	Key Rijndae box Inverse Rijndae box	128 2048 2048	Symkey Rijndae iRijndae
DES	Key	512	Symkey
RSA	Encrypt Key Decrypt Key	192 192	ASymkeyEnc ASymkeyDec
BlowFish	Key	192	Symkey
<common>	Random Seed	128	seed

If these or other data are generated on the client side, the script *tags* can be omitted, but now the script should include itself the generation method. In this case, the amount of required memory should be specified and data should be computed during the initialization stage. According to the OpenCL API, these dynamic structures must be allocated in the **global memory** of the device.

As OpenCL does not provide a simple way to manipulate dynamic variables from the kernels, we propose to create and reference then through a simple interface with an allocation table provided by our architecture. The allocation table has a fixed size, generally of some megabytes, and it's initialized when the OpenCL module starts. We

also include some methods like ‘malloc’ and ‘calloc’ defined in *clmemory.h* header file. These methods can be used from the kernels code.

We consider that memory is split in several memory blocks, each one corresponding to a variable (in this case, the maximum supported are 20 variables) as it is supposed to store simple structures as look-up tables or dictionaries. The allocation table also indicates extra-information, as *ouputSize* and debugging messages to be read from the platform. It can also store user variables, to use between kernels. This version supports only supports *single-thread* allocation.

The structure of the allocation table looks like **Figure 4**.

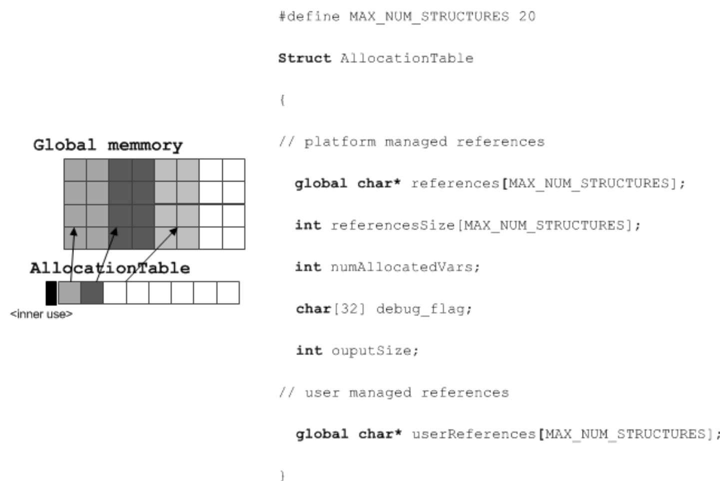

Fig. 4. Allocation Table

Choosing whether to parameterized kernels, on the server or on the client side has its advantages and disadvantages depending on the application. The advantage of initializing structures on *server*-side leads to already runnable scripts, reducing initialization time on the *client*. In turn, as the data are kept in constant memory space, scripts are more efficient. The main disadvantage is that server could be overloaded. Initializing structures on the client side (transmitter and receiver), reduces server overhead, but has some limitations respect to key generation and shared data. In this case, the server should always include some common initialization data (for example a global timer) to be used as seed that is shared by both sides of communication.

4.3 Scripts organization

The scripts should contain at least two methods: *init* and *encrypt*. The *init* method initializes structures in the host memory and it’s called only once, after receiving the key from the server. If script was already instantiated in the server and do not use local variables, this step can be omitted.

The *encrypt* method is called after each new frame issent. The encryption method receives as parameter an input (src) and an output (dst), that could not be the same

variable, the buffer size and the allocation table reference. Below it is the template structure for an encryption 'script' that works both for client-side initialization or server-side initialization.

```
// data initialized on server side
constant word symmetrickey[4]   = {0Xb..... };
// unit for memory allocation.
#include "clmemory.h"

kernel void init(AllocationTable mt)
{
  // host data initialization
   global char* myVariable = malloc( XXX bytes , mt);
   generateStructure( myVariable );
   // Store reference to be accessed from other kernel
   mt->references[0] = myVariable;
}

kernel void encrypt(global char src, global char dst, int
buffersize, AllocationTable mt)
{
  // to access locally initialized structures
  global char* auxStructure = mt->references[0]  ; \\
  // Encrypt Code
   ....
}
```

Finally, the C code-like that implements the whole encryption scheme is presented.

```
/*  SERVER */
onNewRequest()
  ak = chooseAlgorithm() ;
  keys = generateKeys(ak) ;
  replaceTags(ak,  keys) ;
  secureSend(ak) ;
/*   HOST */
// Call once, at the begining
onStartSending
  ak = readAlgorithmFromServer();
  AllocationTable mt  =openCL.initializeAT([Mem Size]);
  openCL.compileKernel(ak);
  //buffer size is equal to frame size
  openCL.allocateBuffer( size );
  // Initialize local structures \\
  openCL.callKernel('init');
```

```
// For each frame
onSending
frame = readFrame();
if (frame)
{
  openCL.copyHostToDevice(frame);
  openCL.callKernel('encrypt');
  openCL.copyFromDevice(outFrame);
  send(outFrame) ; \\
}
```

This scheme could be extended to work like a *pipe and filter architecture*. In this case, all algorithms should implement the same interface and the architecture must call one kernel after the other.

5 Experimental results

Several implementation and performance analysis were performed. First compatibility features running with OpenCL were evaluated. After it, performance tests with different encryption techniques using sequences of still images were conducted, both with CPU and GPU. Here, we only considered encryption/decryption times, omitting the communication times that depend on the application.

We used different algorithms, such as Blowfish, AES, DES and RSA. The test platform, the clients and the server were implemented in C++. For communication, UDP sockets with Boost library were used. Encryption algorithms were implemented in ANSI C for OpenCL and stored and transmitted as plain text. Keys and structures were generated only once, and were used the same in all tests. We used a six-core PC at 3.0 GHz with 4 MB of RAM and a GTX 550 GPU.

5.1 Implementation Analysis

Each device has different OpenCL capabilities: number of cores, threads and memory spaces size. At the same time, there are different versions of this platform: ATI, NVIDIA, Intel among others, that should comply with the standard. As OpenCL still does not support recursion, algorithms have some limitations.

In this analysis, we intend to test the capability of running algorithms in different configurations. For this analysis, we took into account the amount of memory required, the amount of lines of each algorithm, the constant memory space used, the compilation time and the maximum call-stack depth. The algorithms compositions are shown in **Table 2**. We used the OpenCL's NVIDIA version.

Table 2. Table memory spaces and lines of code

Cipher	KeySize	Code Lines	Constant Space	Compilation Time (ms)	Callstack Depth
AES	128 bits	250	844 Kbytes	2.7	3
DES	192 bits	512	1294 Kbytes	3.5	3
BlowFish	256 bits	310	252 Bytes	5.3	2
RSA	128 bits	1200	6 Kbytes	131	8

As expected, compilation times were proportional to the length and complexity of the code. In some cases, compilation times were very high, and depending of GPUs platform (using older GPUs than the one proposed), the RSA algorithm with a "call-stack depth" of 8 o more could not be compiled. It is clear that the greater complexity of the algorithms, the longer the compiling time.

In another test, Firefox and WebCL were used with a plug-in developed by Nokia ®, as shown in **Figure 5**. We implemented a simplified version of the client module; the algorithms were included within the web-page code. The compilation and execution times through the browser were very similar to those obtained in implementing C++ thanks that this step is carried out by the API. On the other side, the data copy times between CPU-GPU were 50% higher.

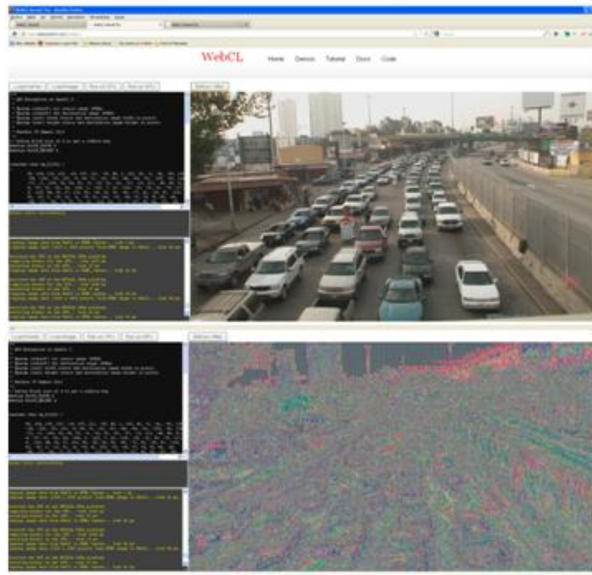

Fig. 5. : Screenshot of *AES* encryption running *in a* WebBrowser

5.2 Performance Analysis

In the following tests, we calculated the rate of processing images measured as Megabits per second (Mbps) obtained both CPU and GPU. We encrypted a sequence

of 30 images in uncompressed format with different resolutions of 1024x640, 1280x800 and 1920x1080 with 3 bytes per pixel. The input data is partitioned into blocks of 8192 bits for parallel processing. **Table 3** summarizes the results of different implementations.

Table 3. : Throughput in Mbps obtained for different algorithms in CPU & GPU

	AES	DES	blowfish	RSA
CPU (6 Cores)	240	144	736	4
GPU	1920	368	8192	20

The obtained results let us affirm that AES and Blowfish can be used in real-time encryption. On the other side, DES and RSA were not fast for multimedia data encryption; even they were running in parallel. Comparing to [4], the obtained throughput of AES is about 5 times slower; but our proposal is more generic as it supports many different algorithms.

5.3 Multi-Step Analysis

Finally, a hybrid encoding-encryption test was performed with a JPEG2000 image encoder in OpenCL. This encoder was implemented *inside* the platform (not transmitted) and applied for each frame. It uses default parameters such as 32x32 pixels block size. As the image is encoded in the GPU, memory swaps are reduced. Starting from a compressed frame with this *codec*, we then applied DES and AES algorithms for encrypting the image. The time required for each step is shown in **Table 4**.

Table 4. Times in milliseconds for encoding + encryption

	Encoding		Encryption		
Image resolucion	**Orig Size**	**Encoding Time (ms)**	**Compressed Size**	**AES (ms)**	**DES (ms)**
1024x640	1912kb	181	844kb	2.7	67
1280x800	3001kb	243	1294kb	3.5	84
1920x1024	6076kb	321	2434kb	5.3	131

In these tests, it was observed that most of the processing is taken by the encoder. Even though it is not a good configuration for real-time, it shows us that the architecture can work as a JVCE scheme.

6 Conclusions

In this paper we presented a new architecture for efficient and reliable transmission of large data volumes. Although originally tailored to video images, the same can be applied to any other domain where encryption algorithms must be chosen according

to the problem. The architecture is still in development and we are evaluating new algorithms and carrying out some analysis in strength against attacks.

Preliminary results are promising. On the one hand, it allowed us to decouple the data structure from encryption algorithms, reducing the vulnerability of the communication channel. At the same time, we obtained a high processing rate thanks to using OpenCL on GPUs for the development. The idea of having algorithms coded in script gives us a greater number of possibilities in the ways of encrypting. As future work, we will explore the dynamic generation of algorithms, from the combination of basis algorithms and we also pretend to extend the architecture to incorporate 3D images, used in medical applications.

7 References

1. Al-Husainy M. F. : A Novel Encryption Method for Image Security, International Journal of Security and Its Applications v. 6:1, pp.1-8 (2012)
2. Daemen J. and Rijmen V. : The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag, ISBN 3-540-42580-2 (2002).
3. Hellman M.: An Overview of Public Key Cryptography, IEEE Communications Magazine, pp:42-49 (2002).
4. Iwai K., Nishikawa N., Kurokawa T.: Acceleration of AES encryption on CUDA GPU, International Journal of Networking and Computing, v. 2:1, pp. 131-145 (2012).
5. McGrew D., Naslund M., Norman K., Blom R., Carrara E. and Oran D.: The Secure Real time Transport Protocol (SRTP), Internet draft, (2001).
6. Meyer J. and Gadegast F.: Security Mechanisms for Multimedia Data with the Example MPEG-1 Video, Project Description of SECMPEG, Technical University of Berlin, Germany (1995).
7. Liu F., Koenig H.:A survey of video encryption algorithms, Computers and Security, v.29:1, pp. 3-15 (2010)
8. Nishikawa N., Iwai K. and Kurokawa T. Acceleration of the key crack against cipher algorithm using CUDA (in Japanese). In IEICE technical report. Computer systems v. 109:168, pp. 49-54 (2009).
9. Pande A. , Zambreno J.: The secure wavelet transform, Journal of Real-Time Image Processing, Springer-Verlag, DOI 10.1007/s11554-010-0165-6 (2010)
10. Pieprzyk J. and Pointcheval D.: Parallel Authentication and Public-Key Encryption , The Eighth Australasian Conference on Information Security and Privacy (ACISP 03), Ed. Springer-Verlag, LNCS 2727, pp.383-401 (2003).
11. Rosenthal A., Mork, P. Li M.H., Stanford J. , Koester D. and Reynolds P. ; Cloud computing: a new business paradigm for biomedical information sharing. Journal of Biomedical Informatics v.43, pp.342-353 (2010).
12. Samid G. , Encryption-On-Demand: Practical and Theoretical Considerations. IACR Cryptology ePrint Archive 2008: 222 (2008)
13. Shin S. U., Sim K. S. and Rhee K. H.: A Secrecy Scheme for MPEG Video Data Using the Joint of Compression and Encryption, 2nd International Workshop on Inf. Security, Kuala Lumpur, Malaysia, Lecture Notes in Computer Science, v. 17, pp.191-201 (1999).
14. Subashini S. , Kavitha V. :A survey on security issues in service delivery models of cloud computing, Journal of Network and Computer Applications, v.34:1, pp. 1-11 (2011)
15. Stinson D.R.: Cryptography Theory and Practice, CRC Press, Inc. (2002).