Encrypting video and image streams using OpenCL code on-demand

Juan P. D'Amato
Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA)
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET),
Facultad de Ciencias Exactas,
Tandil, Argentina, 7000
jpdamato@exa.unicen.edu.ar

and

Marcelo J. Vénere
Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA)
Comisión Nacional de Energía Atómica (CNEA),
Facultad de Ciencias Exactas,
Tandil, Argentina, 7000
venerem@exa.unicen.edu.ar

Abstract

The amount of multimedia information transmitted through the web is very high and increasing. Generally, this kind of data is not correctly protected, since users do not appreciate the amount of information that images and videos may contain. In this work, we present architecture for managing safely multimedia transmission channels. The idea is to encrypt or encode images and videos in an efficient and dynamic way. At the same time, these media could be enhanced applying a real-time image process. The main novelty of the proposal is the application of on-demand parallel code written in OpenCL. The algorithms and data structure are known by the parties 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: Encryption, GPU, on-demand, OpenCL.

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 radical change in the way of delivering and accessing data. Today, there are many applications in different 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; since they have been designed for generic uses. They also do not consider the different user processing capabilities and even worst, they are not prepared for the new ways of storing data such as clouds [14].

In a virtualized storing and computing model where the data host is not the data owner; both data and application that manage this data can both be expropriated. A critical case is imaging repositories (known as PACS) that have

migrated to the cloud [11]. These repositories keep important confidential information of patients in different image formats that is shared and accessed by many users in many different ways. In this context, a new method to ensure data security and adaptable visualizing methods is essential.

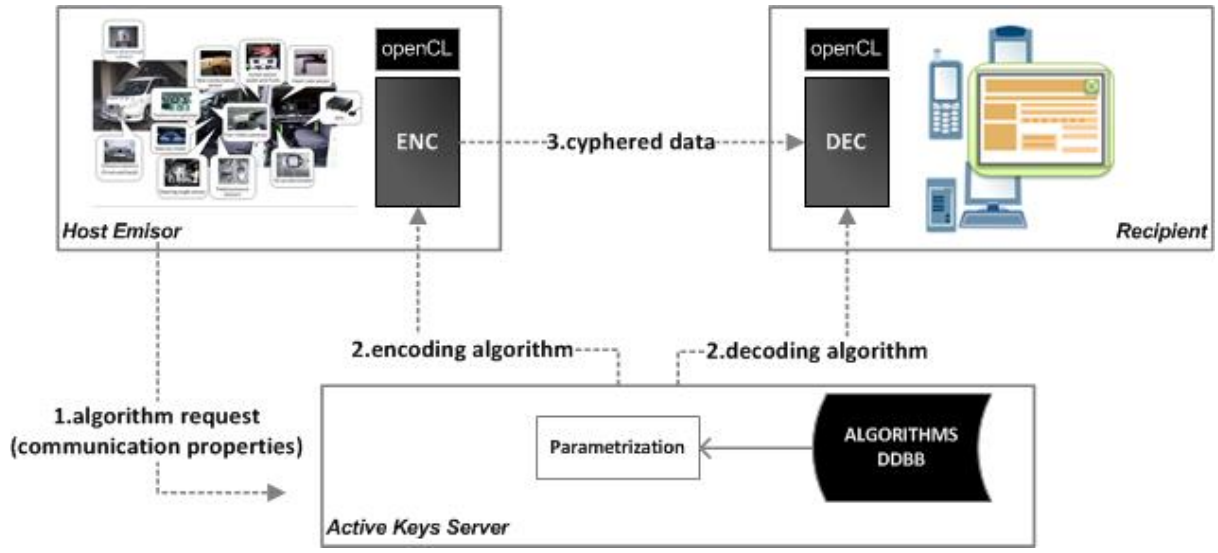


Figure 1: Proposed Architecture

These performance, distribution and adaptability to different configurations issues have motivated our proposal. The idea is to develop a platform that dynamically selects 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. The layout of the architecture is shown in (Fig. 1).

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 an enriched strategies database. Throughout the paper, we present the proposed architecture and some performance results obtained from image processing. We think that an security analysis was not essential, as we are using 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 *'naïve'*. 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). There exists a commercial SDK that claims to obtain 10GB per second using AES[18], far enough for processing real time high definition video.. 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. In [16], he proposes that the main goal is to develop algorithms that trends to the optimal point indicated in the (Fig. 2) considering three factors: coding efficiency, Power efficiency and Privacy.

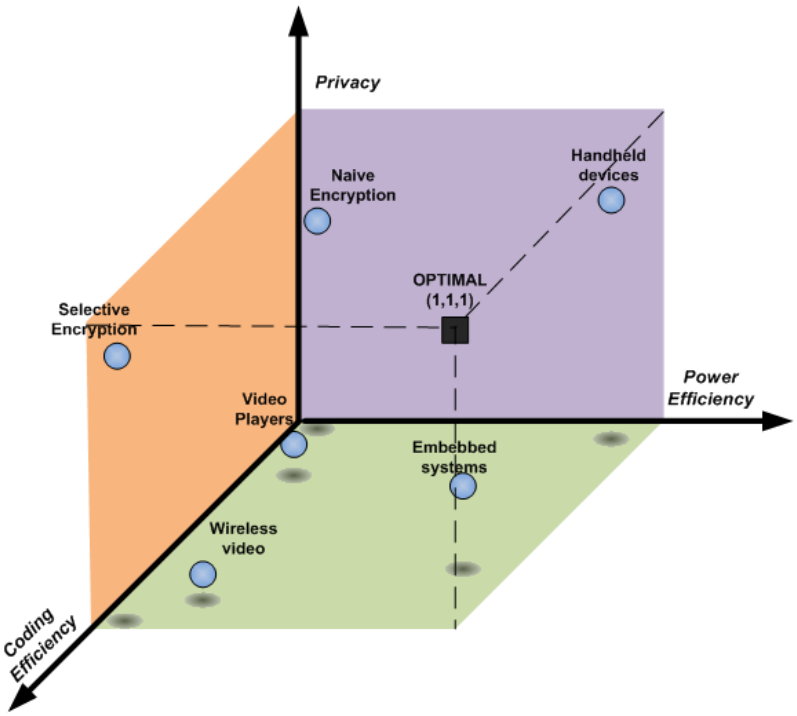


Figure 2: The key features of multimedia encryption

Considering this classification, one of the issues that 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]. This paradigm where graphic cards are used in different kind of problems is called General Purpose Computing on Graphics Processing Units or GPGPU.

The proposed solutions to video encrypting or encoding generally often propose NVIDIA CUDA as a development platform, as can be found in [18] . This platform is very complete and stable, with many development tools but its application is limited to personal computers (desktops or clusters). Another alternative to achieve parallelism on different platforms is the standard OpenMP. This API runs on Linux, Windows, iOS and Android, but it has the limitation of using only pre-compiled code, and it has been only ported to C and FORTRAN.

The third upcoming 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. 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 (Fig. 3).

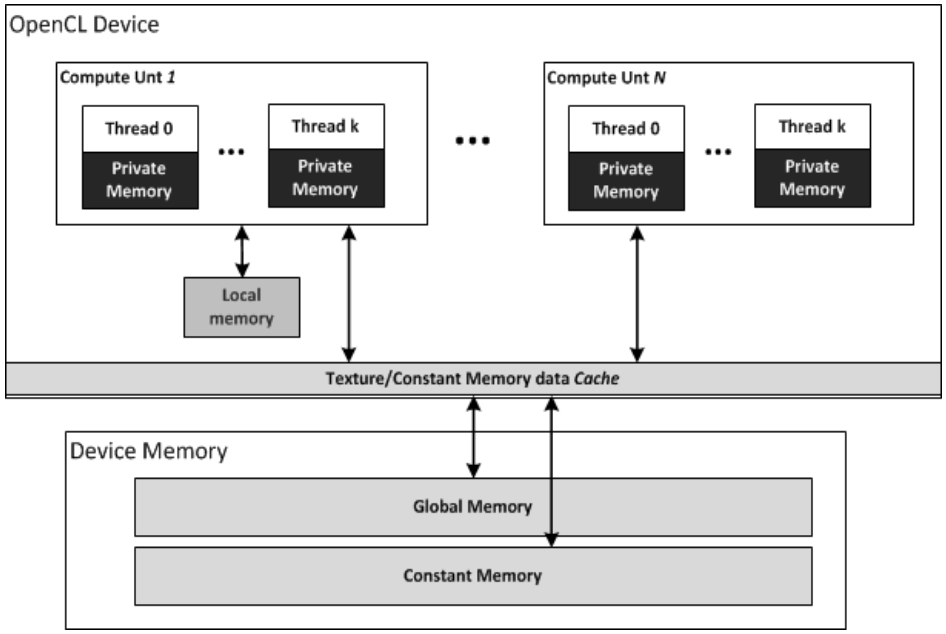


Figure 3: OpenCL memory scheme

One of the main features of OpenCL is that code is loaded dynamically and compiled into the GPU (or device) at running time. 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 over 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 provider, 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.

The client should be capable of processing OpenCL code. For this purpose, there is a client module that can load, compile and execute OpenCL kernels. This module can also interchange data between the host and the device. The incoming data, e.g. a frame from a video, are loaded and processed in the module before sending through back 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 (Fig. 4).

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 video encryption, the same schema can also be applied to compress or process videos and pictures in parallel.

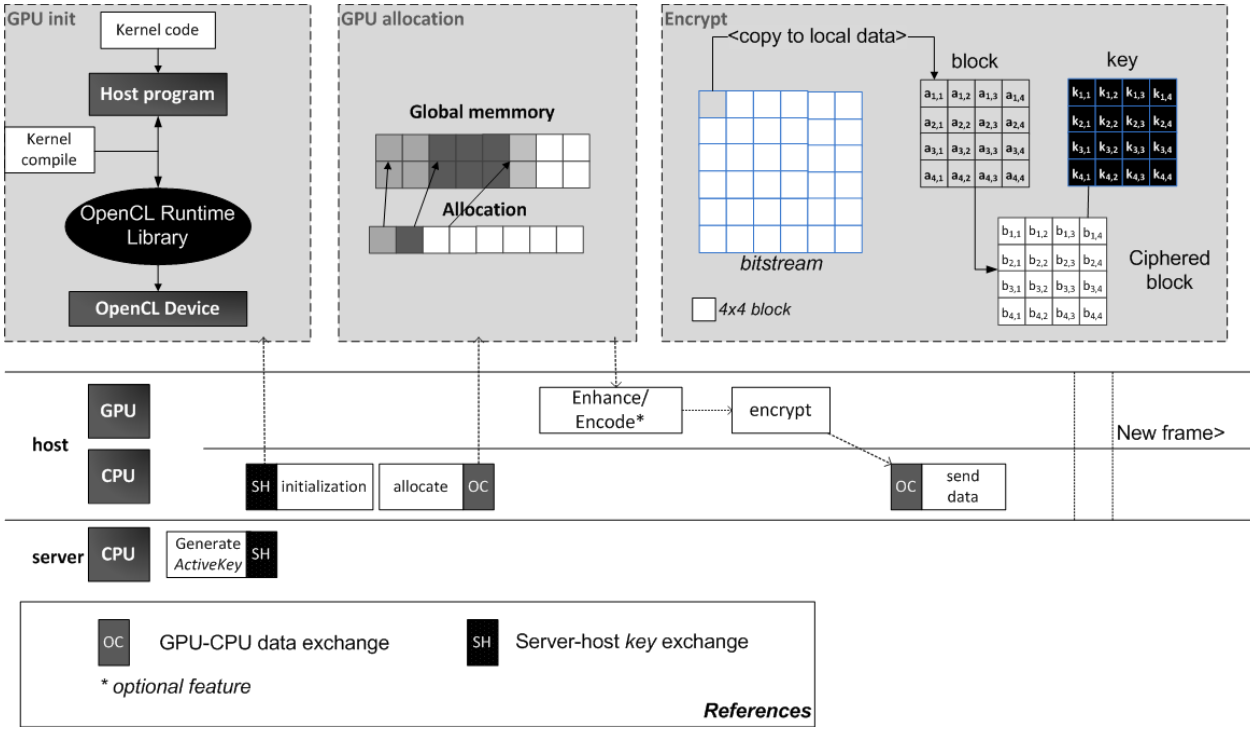


Figure 4: Server-host encryption scheme

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 both the key and 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 if new algorithms are included, such as the new one proposed in [1]. Thanks to this improvement, at request time, the server can choose among plenty of algorithms, reducing vulnerability to attacks. At the same time, the techniques must be implemented following a certain structure. For this purpose, we also propose a code template and a mechanism for describing auxiliary structures; both explained 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 a range exploits. To ensure that algorithms incorporated to the database are valid, they must be exposed to a series of tests with different data sizes.

4.2 Keys and structures generation

One of the most important steps before encryption is the algorithms parameterization. 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], dictionary 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 structures be must be included in the algorithm script itself as a constant with values from a base type (char or integer). When client receives the script, it compiles and runs it.

To let the server automatically generate and 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.

With this approach, many clients could encrypt and decrypt the same video at the same time without extra-communication. Even more, the initialization in the client is quite fast and light, as code only includes the encrypt functions.

Table 1 : Algorithms parameters with their corresponding *tags*

Cipher	Parameters	Size (Bits)	Tag
AES	Key	128	Symkey128
	Rijndael box	2048	Rijndael
	Inverse Rijndael box	2048	iRijndael
DES	Key	512	Symkey512
RSA	Encrypt Key	192	ASymkeyEnc192
	Decrypt Key	192	ASymkeyDec192
BlowFish	Key	192	Symkey192
<common>	Random Seed	128	Seed128
	Global Timer	64	GTimer64

If these or other data are generated on the client side, many of the script tags can be omitted, but now the script should include itself the generation method. For instance, if a client requires a 128 bits key, it must be created during initialization stage. In this case, the amount of required memory used by the GPU should be somehow specified.

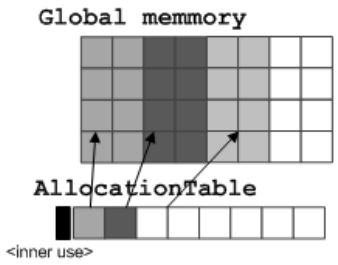
As OpenCL does not have dynamic allocation methods, we must develop a strategy that let us generates this structures in a simple way. As shown in the section 2, these structures must be allocated in the global memory of the device, which is a bit slower than local, but it can be accessed by all threads in the device.

4.2.1 Pseudo-dynamic memory management

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). It is supposed to store simple structures as lookup tables or dictionaries. The allocation table also indicates extra information, as debugging messages to be read from the platform. It can also store user variables, to share between kernels. The actual version of the architecture only supports only single-thread memory allocation, task that is done during initialization. The structure of the allocation table looks like (Fig. 5).

Choosing whether to parameterize 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.




```
#define MAX_NUM_STRUCTURES 20
Struct AllocationTable
{
// platform managed references
  global char* references[MAX_NUM_STRUCTURES];
  int referencesSize[MAX_NUM_STRUCTURES];
  int numAllocatedVars;
  char debug_flag[32];
  int ouputSize;
// user managed references
  global char* userReferences[MAX_NUM_STRUCTURES];
}
```

Figure 5: Allocation Table proposed for managing variables in GPU

4.3 Scripts organization

The scripts should contain at least two methods: init and encrypt. The *init* method initializes structures in the host memory. It is a single *thread* call and it is called 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 is sent. 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. Data is declared using a generic type, as char Pointer, and it should be instantiated when used. This approach was done because of limitations of actual OpenCL version.

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(X size in 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, Allo-
cationTable* 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 */
// Called 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)
{
// Data is copied to device
  openCL.copyHostToDevice(frame);
// Kernel is invoked and run
  openCL.callKernel('encrypt');
//Result is read back to host
  openCL.copyFromDevice(outFrame);
  send(outFrame) ;
}
```

This scheme is extended to work like *pipe's and filter's 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 combined with image enhancement techniques using sequences of still images and videos were conducted, both with CPU and GPU. We used different algorithms, such as Blowfish, AES, DES and RSA, implemented for this work considering performance issues. The implementations run at similar rates the best known implementations.

For the test, we developed a client in C++ and other in HTML5. The server algorithms were implemented by now in C++. Encryption algorithms were implemented in ANSI C for OpenCL and just for testing purposes they were stored and transmitted as plain text. Keys and structures were generated only once, and were used the same in all tests. We used a AMD FX 6100 six-core PC at 3.0 GHz with 4 GB of RAM and a NVIDIA 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, and 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 KiB	2.7	3
DES	192 bits	512	1294 KiB	5.3	3
BlowFish	256 bits	310	252 Byte	3.5	2
RSA	128 bits	1200	6 KiB	1031	8

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

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 640x480, 1280x800 and 1920x1080 with 3 bytes per pixel, with a size of 1, 3 and 6 Megabytes respectively. A real time video requires about 25 frames per second.

The input data is partitioned into blocks of 8192 bits for parallel processing. (Table 3) summarizes the best results of different implementations.

Table 3. : Best 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
SpeedUp	8x	2.5x	11.13x	5x

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 enough for multimedia data encryption; even they were running in parallel.

Comparing to the best solution of [4], the obtained throughput of AES is about 5 times lower and could be improved; but our proposal is more generic as it supports many different algorithms.In (Fig. 6), we show how the amount of frames that are processed per second decreases as image size increases. Here it's shown two algorithms with two implementations (CPU and GPU). *Mbps* is equal to *FPS * ImageSize in Bits.* It's observed that with images bigger than 1280x800, the GPU starts to give a better throughput.

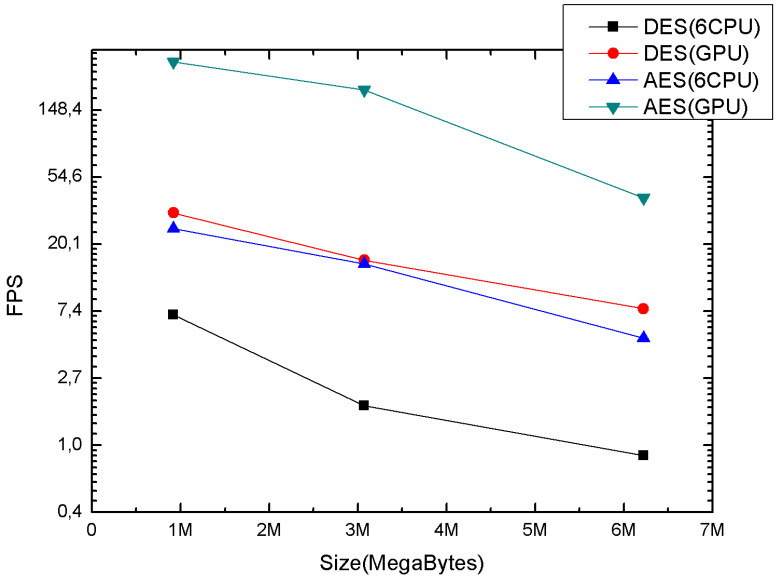


Figure 6: Amount of data that can be encrypted per second.

5.3 Multi-Step Analysis

Then, 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 is known that JPEG2000 (a.k.a. J2K) is hard to implement in parallel, and many works treat this problem as [20].

Our implementation works at a reasonable speedup respect to its CPU version and the best known J2K encoders [19] (about 6 times faster). It uses default parameters such as 32x32 pixels compression block size. As any compression schemes, memory allocation should be done dynamically. This dynamic memory management on GPU is solved with the approach presented in section 4.2. At the same time, as the image is encoded in the GPU, memory swaps are reduced and general performance is kept high.

The idea is starting from a compressed frame with the J2K *codec* and then applied DES and AES algorithms for encrypting the image. Images are in the same resolution as previous tests. The time required for each step for each image size is shown in (Table 4).

Table 4: Times in milliseconds for encoding + encryption

	Encoding		Encryption		
Image resolution	**Orig Size**	**Encoding Time (ms)**	**Compressed Size**	**AES (ms)**	**DES (ms)**
640x480	921KiB	181	844kb	3.7	31
1280x800	3001KiB	243	1294kb	13.5	72
1920x1024	6076KiB	321	2434kb	25.0	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.

5.4 Web Browser Client

In another test, Firefox and WebCL were used with a plug-in developed by Nokia [17]. This plugin is still under development, and it not fulfills the OpenCL specification. We implemented *JavaScript* version of the client module running in HTML5, as can be seen working in (Fig.7). The algorithms are customized and stored in the HTML code in the server and dynamically loaded when the web-page is created.

This test page has several function that user can change. First, the data source (local or remote image file, camera or video stream). Then, user can also select the OpenCL compatible platform to runs the test, either CPU or GPU. And finally, it includes the list of available scripts and some monitors to evaluate the use as amount of memory Device or time required for processing. In this case only the DES algorithm is used. The test was done on some medical images.

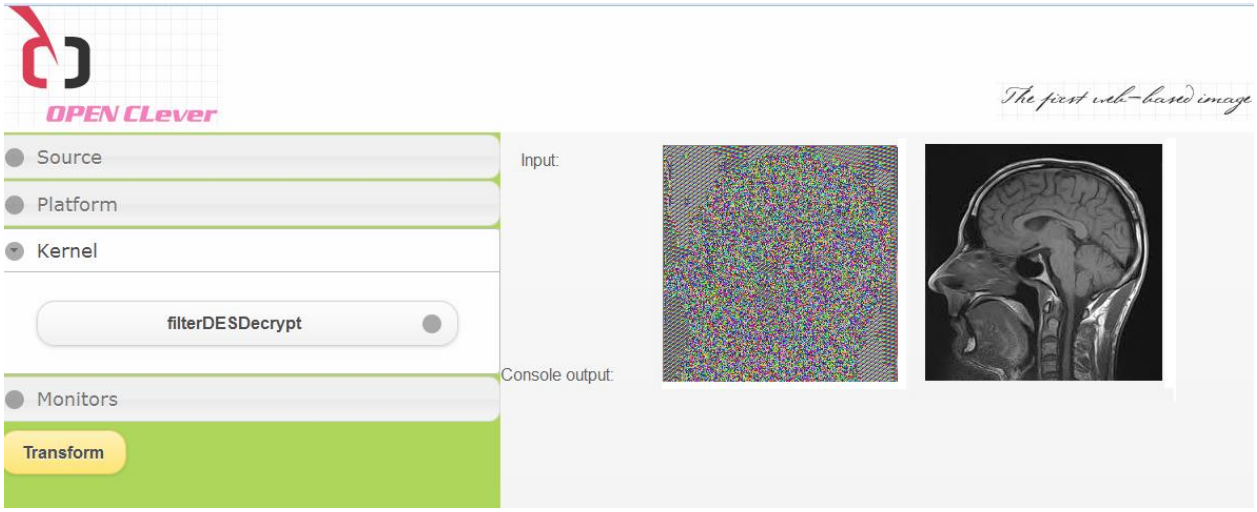


Figure 7: Screenshot of AES decryption running in a WebBrowser. (left) the user panel control (right) the original and transformed image are shown.

The compilation and execution times through the browser were very similar to those obtained in the C++ implementation thanks to execution step is carried out entirely in the Device (in this case, a GPU). On the other side, the data transfer times between CPU-GPU were 50% higher. This overhead comes when read data is adapted to Javascript structure and then rendered to a HTML Canvas. In the future, the idea is to use WebGL and device interoperability feature, to direct render data to the screen, avoiding the adaptation step.

6 Conclusions

In this paper, a new architecture for efficient and reliable transmission of multimedia data was presented. The idea of having algorithms coded in scripts brings us a great number of possibilities in the ways of encrypting. The architecture is still in development and we are evaluating new algorithms and carrying out some analysis in strength against attacks.

Current 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 and what it is more important, algorithms can be selected according the platform requirements. At the same time, we obtained a high processing rate thanks to OpenCL API. The WebCL client version let us run the samples in different platforms. The idea is to provide a visualization solution that could run in the future on tablets and smartphones with OpenCL compatibility. The main problem is that OpenCL API implementations in different Operative systems are rather old, incomplete and buggy. It has not all the types and functions defined in the last Khronos Specification. Sometimes scripts should be rewritten in order to get them work in all the devices. This technical limitation should be avoided in the near future, once the API matures.

Although, this platform was originally prepared for video and image processing, the same idea can be applied to any other domain where adaptive encryption algorithms must be chosen. 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.

Acknowledgments

This work was partially founded with the PICT 1287/11 called “Procesamiento y segmentación de imágenes digitales tridimensionales para el desarrollo de aplicaciones médicas e industriales” from the MinCyT (Ministerio de Ciencia, Tecnología e Innovación Productiva).

References

[1] M. Al-Husainy : A Novel Encryption Method for Image Security, International Journal of Security and Its Applications v. 6:1, pp.1-8 , 2012.

[2] J. Daemen and V. Rijmen: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag, ISBN 3-540-42580-2, 2002.

[3] M. Hellman: An Overview of Public Key Cryptography, IEEE Communications Magazine, pp:42-49 , 2002.

[4] K. Iwai , N. Nishikawa, T. Kurokawa: Acceleration of AES encryption on CUDA GPU, International Journal of Networking and Computing, v. 2:1, pp. 131-145, 2012.

[5] D. McGrew, M. Naslund K. Norman K., R. Blom, E. Carrara and D. Oran: The Secure Real-time Transport Protocol (SRTP), Internet draft, 2001.

[6] J. Meyer and F. Gadegast: Security Mechanisms for Multimedia Data with the Example MPEG-1 Video, Project Description of SECMPEG, Technical University of Berlin, Germany, 1995.

[7] F. Liu , H. Koenig:A survey of video encryption algorithms, Computers and Security, v.29:1, pp. 3-15, 2010

[8] N. Nishikawa, K. Iwai and T. Kurokawa: 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] A. Pande , J. Zambreno: The secure wavelet transform, Journal of Real-Time Image Processing, Springer-Verlag, DOI 10.1007/s11554-010-0165-6 , 2010.

[10] J. Pieprzyk and D. Pointcheval : 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] A. Rosenthal, P. Mork, M. Li, J. Stanford, D. Koester and P. Reynolds: Cloud computing: a new business paradigm for biomedical information sharing. Journal of Biomedical Informatics v.43, pp.342-353, 2010.

[12] G. Samid, Encryption-On-Demand: Practical and Theoretical Considerations. IACR Cryptology ePrint Archive 2008: 222 , 2008

[13] S. Shin, K. Sim and K. Rhee: 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] S. Subashini, V. Kavitha: 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] D. Stinson: Cryptography Theory and Practice, CRC Press, Inc., 2002.

[16] A Pande, P Mohapatra and J Zambreno, Securing Multimedia Content Using Joint Compression and Encryption. Multimedia IEEE, v.99. 10.1109/MMUL.2012.29, 2012.

[17] Nokia WebCL Plugin URL = http://webcl.nokiaresearch.com, 2014.

[18] GKrypt. URL = http://www.gkrypt.com , 2014.

[19] J2K official site. URL = http://j2k.sourceforge.net/, 2002

[20] J. Franco, G. Bernab, J. Fernndez, and M. Acacio: A parallel implementation of the 2d wavelet transform using CUDA. Parallel, Distributed, and Network-Based Processing, Euromicro Conference. DOI 10.1109/PDP.2009.40, 2009.