GPU-Accelerated Password Cracking of PDF Files

Keonwoo Kim[1], Sangsu Lee[1], Dowon Hong[1] and Jae-Cheol Ryou[2]
[1] Information Security Research Division, ETRI
Daejeon, 305-700 - KOREA
[e-mail: wootopian@etri.re.kr]
[2] Department of Computer Engineering, Chungnam National University
Daejeon, 305-764 - KOREA
[e-mail: jcryou@cnu.ac.kr]
*Corresponding author: Keonwoo Kim

Received May 11, 2011; revised September 6, 2011; accepted September 28, 2011; published November 29, 2011

Abstract

Digital document file such as Adobe Acrobat or MS-Office is encrypted by its own ciphering algorithm with a user password. When this password is not known to a user or a forensic inspector, it is necessary to recover the password to open the encrypted file. Password cracking by brute-force search is a perfect approach to discover the password but a time consuming process. This paper presents a new method of speeding up password recovery on Graphic Processing Unit (GPU) using a Compute Unified Device Architecture (CUDA). PDF files are chosen as a password cracking target, and the Abode Acrobat password recovery algorithm is examined. Experimental results show that the proposed method gives high performance at low cost, with a cluster of GPU nodes significantly speeding up the password recovery by exploiting a number of computing nodes. Password cracking performance is increased linearly in proportion to the number of computing nodes and GPUs.

Keywords: Password cracking, GPU, acceleration of password recovery, brute-force, PDF

This work was supported by the IT R&D program of MIC/KEIT. [10035157, Development of Digital Forensic Technologies for Real-Time Analysis]. And, the fourth author of this research was supported by the MKE(Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2011-(C1090-1131-0005))

DOI: 10.3837/tiis.2011.11.021

1. Introduction

From the viewpoint of computer forensics, an encrypted document file may potentially contain a wealth of information about users and their activities. This information can have great value as an evidence in a criminal investigation [1][2]. However, computer files are easily encrypted by user passwords to protect their contents from unauthorized access. When a suspect does not reveal his password or a user does not remember it, encrpted files can be decrypted by password cracking, which is the process of recovering passwords from encrypted files. It might be of help to a forensic inspector to gain access to encrypted files or a user to recover a forgotten password. The operating system stores user passwords in an encrypted hash form. Unlike an OS log-in password to authenticate a user, a user password in applications such as Adobe Acrobat and MS-Office are actually used to encrypt files [3].

The use of a password dictionary [4] is one of the fastest ways to recover a password. Generally, many people have a tendency to choose their passwords as human-memorable words like using their personal information or appending a digit to the information. A part or whole of the password may be found in the pre-defined dictionary. To discover a password depends entirely on a vast dictionary of common password phrases, which can crack simplistic passwords like 12345 and similar patterns. However, there is very little possibility to crack password using a password dictionary if anyone uses a mix of lowercase and uppercase letters, digits and symbols as a password. Making an effective password dictionary including randomized passwords is another challenge.

Rainbow tables are usually used in recovering a password in a short time as storing long lists of pre-computed password hashes. It is a form of Time-Memory Trade-Off (TMTO) [5], using less CPU at the cost of more storage. This method requires a considerable amount of preparation time, but it allows the password recovery to be executed faster. However, as the size of passwords grows, it will take as much time to make the table as the brute-force cracking time. Moreover, not all random passwords can be found by hashes of tables. The password recovery process of common applications includes other types of cryptographic operations and requires many repeated computations of the same algorithm. Therefore, the rainbow table attack might be suitable for cracking a password with a cryptgraphic algorithm like a Data Encryption Standard (DES), not for the password recovery with two and more algorithms. This approach is also partially effective to recover passwords.

On the contrary, a brute-force cracking [6] is a complete method to discover a password. It simply tries all candidate passwords, every possible character combination, until it hits the right one. In the worst case, this would involve traversing the entire search space. The time elapsed to find a correct password by brute-force is related to password length and complexity. Longer password increases exponentially the number of candidate passwords to be checked. Also, using various character types for a password, such as adding uppercase characters, numbers and symbols other than lowercase letters, significantly increases the time required to decipher a password. Therefore, we need to accelerate brute-force cracking in order to search all available passwords space. Most password cracking toolkits are software applications using only CPU computing ability. The performance limit of these tools to achieve a goal is obvious because CPU works as a data cache and a control processor and also in data processing. In order to reduce password cracking time, hardware acceleration can increase the number of candidate passwords per second which must be checked.

This paper proposes a new method for speeding up password cracking on Graphic Processing Unit (GPU) to parallelize the search. The compute-intensive aspect of the task is performed through data parallelism of many processor cores on the chip. GPU is already installed in most computers so that additional cost for password recovery is not a big problem. Our cracking target is an Adobe Acrobat PDF application. We assign a task to run the sequential part of the recovery step on the CPU and the parallel computational part including cryptgraphic operations on the GPU. To try all combinations of available letters, we apply an exhaustive password search, not a password dictionary attack and a rainbow table approach. The experiment by using one GPU and multi GPUs is compared with the software approach.

In the remaining part of this paper, we introduce a password recovery technique that uses a cluster of GPU nodes to decrease the search time of a very long password. The proposed method is proved to be the most efficient way to make full use of all computing devices without having the same candidate paswords between nodes. We provide an estimated cost of the equipment to give an idea on the resources required for successful cracking of long passwords.

2. Related Work

2.1 Hardware Acceleration of Brute-force Password Cracking

Hardware acclerators can be used to greatly speed up brute-force password cracking. Password recovery process of encrypted files usually includes many repeated cryptographic operations. Exelllent capabilities of hardware acceleration have been already proved in the fields of cryptographic analysis and in the security domain.

First, the graphic processor is used to speed up brute-force password recovery over genaral processor. GPU as a coprocessor accelerates cryptographic computing [7][8][9][10][11][12] such as block cipher and hash function as well as graphic processing. A commercial product claims an ability to test up to 8,400 passwords a second using GTX 295 to recover MS-Office 2007 password [13]. And, the recovery process of an eight-character Windows Vista logon password with Windows NT LAN Manager (NTLM) hashing would take only three to five days by GPU acceleration, while a modern dual-core PC searches about 55 trillion possible passwords, tries up to 10 million passwords per second, and performs a complete analysis in about two months [13]. Besides acceleration of password cracking, GPU is used in speeding up Secure Socket Layer (SSL) in SSLShader [14], and outperforming the exsiting software routers [15]. And, fast GPU-based implementaions of the brute-force K-nearest neighbor search algorithm are shown by [16], and a parallel fuzzy connected image segmentation algorithm with GPU is presented by [17]. Representative GPUs that can be used for general operations such as password cracking are Nvidia Geforce, Tesla, and Quadro series.

Second, password cracking can be significantly enhanced by a dedicated cracker such as an Application Specific Integrated Circuits (ASIC) acceleration chip or a Field-Programmable Gate Array (FPGA)-based machine. The dedicated ASIC chip for password cracking can give an answer within minutes, while a software toolkit may take a few years for the same target. However, using the ASIC chip is not cost effective since the chip fabricated to achieve a special purpose of password recovery cannot be utilized in other areas. Alternative approach to commercially available hardware system of ASIC cracker is to use FPGA, which satisfies both high performance and relatively low cost for brute-force password search. The Electronic Frontier Foundation (EFF) built a dedicated password cracker using FPGAs in 1998 and broke a DES 56-bit key in 56 hours, testing over 90 billion keys per second [18][19]. In 2008, the

copacobana rivyera machine reduced the time to break DES in less than a day, using 128 Spartan-3 5000's [20][21]. And, the copacobana introduced efficient implementations of more complex cryptanalysis on asymmetric cryptosystems, which are elliptic curve cryptosystems and number cofactorization for RSA algorithm [22].

Third, the architecture of a Cell processor [23] makes it better suited to hardware-assisted brute-force password cracking. A Cell chip consists of one general processor called a Power Processing Element (PPE) and eight parallel coprocessors called Synergistic Processing Elements (SPEs). As a practical cracking, a PlaySation3 (PS3) gaming console with one chip shows an improved ability compared with a CPU-based machine in the similar price range. PS3, which is turned into a Crackstation, works up to 1.4 billion cycles per second as implementing common ciphers and hash functions using vector computing [24]. This speed boost is attributed to SPEs, each one of which can be effectively trying passwords at the same time.

Fourth, a Tableau TACC 1441 hardware accelerator yields 6 to 30 times gain in performance over CPU. It accelerates the decryption of MS-Office 2007 at a speed of 3,201 passwords a second, while Intel core i5 750 CPU searches only 764 passwords [25]. Connecting multiple accelerators together results in increased performance but this hardware is less cost effective compared with GPU and FPGA accelerator.

2.2 GPU and CUDA

Until a few years ago, to solve non-graphics problem, users had to be knowledgeable about graphics programming languages like OpenGL or Direct3D and had to couch their problems in graphical terms. However, Nvidia has introduced a parallel programming model and software environment called Compute Unified Device Architecture (CUDA) [26] to make possible general purpose computation on its graphic chipset. Even if developers who want to make password cracking program using a GPU do not know graphical languages and hardware at all, they can get a result on the CUDA platform with the standard programming language such as C/C++.

GPU device is implemented as a set of multiprocessors. For example, Geforce 9800 GTX [27] has 16 multiprocessors, and a multiprocessor contains 8 thread processors. At any given clock cycle, each processor executes the same instruction but operates on different data. A multiprocessor has 16KB readable/writable on-chip shared memory. Access to device memory is not often by caching frequently used data into the shared memory.The global memory, constant memory and texture memory of the device can be read from or written to by the host and are persistent across kernel launches by the same application [26]. Besides a card-type GPU, the server-type GPU system can be equipped with a host. Tesla S1070 [28] is a server-type parallel computing system with 960 cores in a unit chassis, and it is designed to solve computing challenges more quickly.

GPU code is written by an extension, declaration, and execution configuration defined in CUDA and the code is distinguished from CPU code. The described code is separately compiled for a CPU host and a GPU device by the CUDA compiler. A portion that is executed many times, but independently on different data, can be isolated into a function that is executed on the device with many different threads [26]. A program on a GPU mapped from the compute-intensive function of a CPU is called a kernel. A kernel operates on arrays of multitudes of blocks called grid, and each block has thread execution units. All threads are scheduled by a thread execution manager. In the case of 9800 GTX, 12,288 threads can be

executed at the same time. Every thread is identified by its blockID and threadID, which are a block number within a grid and a thread number within a block.

3. Password Recovery of an Encrypted PDF File

All strings and streams of a PDF file are encrypted by a "Document Open Password" and/or a "Permissions Password". An open password is used both to prevent reading the document by an unauthorized user and to decrypt the encrypted file by an authorized user. To edit and print, the document is restricted by a permissions password. We have to know the open password to open an encrypted file. In order to recover the open password, two password verification methods can be used: one that uses the Ovalue algorithm and the other that uses Uvalue algorithm.

First, an open password can be recovered by computing and comparing Ovalue. **Fig. 1** shows the process of Ovalue computation. Security entities such as Ovalue, Uvalue, IDvalue, and Pvalue must be obtained from the file in advance before calculating Ovalue. A guessed open password is used as input plaintext of RC4 stream cipher [29] to output Ovalue. A guessed permission password is used as input of MD5 hash function [30] to make an encryption key of RC4. If the permissions password is not chosen by a user, the open password is used to substitute it.

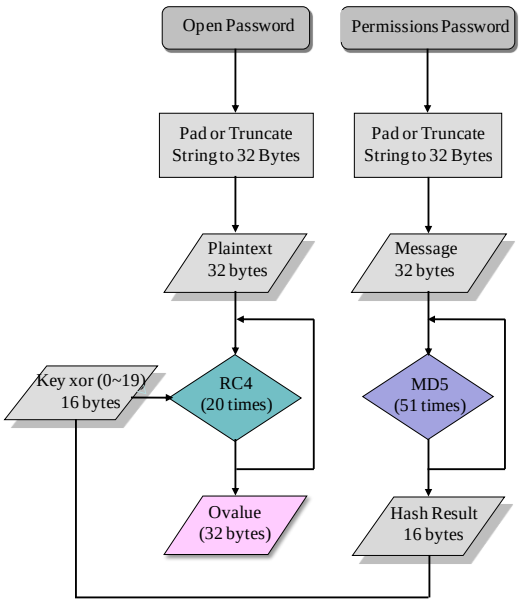

Fig. 1. Ovalue computation process of a PDF file

Ovalue algorithm of the box below describes the flow chart in **Fig. 1** more specifically. By Ovalue algorithm, we can see that the guessed candidate password is not a right open password if the calculated Ovalue is not the same as the obtained Ovalue. This procedure continues using a new guessed password until two Ovalues agree with each other.

[*Ovalue algorithm*]

1. A guessed permissions password should be exactly 32 bytes. If the password string is less than 32, pad it by appending the following PDF padding data.
   28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08
   2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A
   If the length is more than 32 bytes long, use its first 32 bytes.
2. After initializing MD5 hash function, 32-byte data of step 1 become input of this function. Hashed value is calculated for the first time.
3. Following procedure is repeated 50 times.
   A. Previous hashed value becomes input of next hash function. First input of this step is the result of step 2.
   B. Process the hash function.
4. The most significant 16-byte data of the final MD5 output in step 3 are used as an encryption key of RC4 cipher.
5. A guessed open password is padded or truncated to make it 32-byte like step 1. Padded 32-byte data are input plaintext of RC4 stream cipher.
6. Input plaintext of step 5 is encrypted using the encryption key of step 4.
7. Following procedure is repeated 19 times.
   A. Output ciphertext of previous RC4 cipher becomes input plaintext of next RC4. First output of this step is the result of step 6.
   B. New encryption key for this step is changed every time by Exclusive OR (XOR) operation of the encryption key of step 4 and iteration times.
8. The final result of step 7 is a calculated Ovalue.

Second, an open password can be recovered by generating and comparing Uvalue as in **Fig. 2**. An open password as a candidate password is input to generate Uvalue with Ovalue, Pvalue, and IDvalue. The correctness of the candidate password is verified by comparing the calculated Uvalue with the obtained Uvalue. We present Uvalue algorithm to explain the flow chart in **Fig. 2** in greater detail.

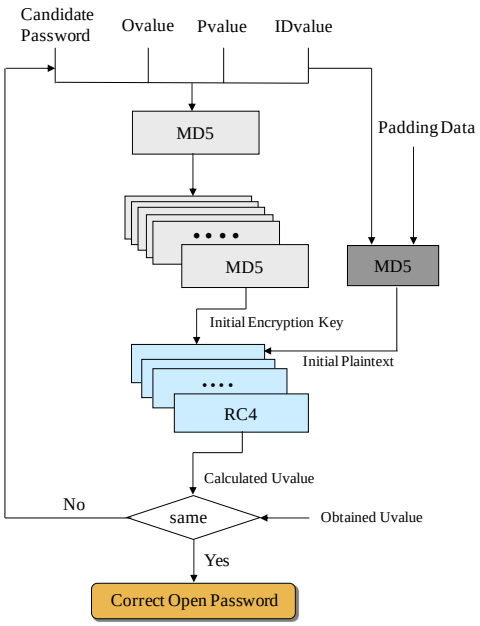

Fig. 2. Password recovery process by generaing and comparing Uvalue

[*Uvalue algorithm*]

1. The size of a candidate password should be 32 bytes. If the size is not 32 bytes, pad or truncate string to 32 bytes.
2. The candidate password is hashed by MD5 hash function with obtained 32-byte Ovalue, 4-byte Pvalue and 16-byte IDvalue.
3. The hashed result of step 2 becomes the input of second MD5. And, the second hashed result is a new input of next MD5 again. This procedure is repeated 49 times. The final hashed value is an initial encryption key of RC4 stream cipher.
4. Hashed value of 32-byte padding data and 16-byte IDvalue is the first input of RC4 stream cipher.
5. The RC4 encryption process is repeated 20 times. The result of step 4 as input data is first encrypted using the result of step 3 as an encryption key. Previous RC4 output stream is used as the input data of next RC4. Key buffer value used in each encryption process is changed every time. The last RC4 output stream is a calculated Uvalue.
6. If the calculated Uvalue is equal to the obtained Uvalue, the candidate password is proven as a correct open password. Otherwise, new candidate passwords are replaced one by one, and the same consecutive process is repeated until two Uvalues are identical.

To open a file encrypted by both an open password and a permission password, a recovery algorithm by Uvalue comparison is preferred to Ovalue algorithm. We should guess not only an open password but also a permissions password if Ovalue algorithm is used. A document secured only by a permissions password opens without an open password.

Table 1. Cracking algorithm to recover an open password of PDF files

Open password	Permissions password	Open	Algorithm to recover an open password
O	O	X	Uvalue
X	O	O	X
O	X	X	Ovalue, Uvalue

4. Acceleration of Password Cracking on GPU

There is a limit on performance to recover a password through a sequential search of a general processor. Our password cracking approach uses both CPU and GPU to boost the performance. In this section, we propose a method of accelerating password recovery on GPU by using Uvalue algorithm. Geforce 9800 GTX [27] and four Tesla C1060s [31] were chosen for our test.

4.1 Password Search Using GPU

Speeding up password search on GPU needs to avoid overlapping work between a host CPU and a device GPU. **Fig. 3** shows the password cracking job to be performed at a host and a device. Security entities of PDF files are obtained from the host and copied into a constant memory of the device. The host generates candidate passwords and one of the candidates is the right password as a result of accelerating password search on GPU. Usable character types as passwords are lower/upper case alphabet letters, arabic numerals, and special characters. Therefore, all passwords consist of combinations of arbitrary length driven from 95 characters and the number of available candidate passwords is calculated as:

$$N = \frac{a(r^{l_m} - 1)}{r - 1} - \frac{a(r^{l_n - 1} - 1)}{r - 1} = \frac{a(r^{l_m} - r^{l_n - 1})}{r - 1} \quad (1)$$

where N is the number of overall candidate passwords to be checked, a is the number of one-character password at selected character types, r is the number of characters usable as passwords of the type, and, l_m and l_n are the expected maximum and minimum length of a password. a and r are of the same value at any selected character types.

Every candidate password is 32-byte long. The number of candidate passwords to be generated at a time is identical to the number of GPU threads that are used to accelerate password recovery, and calculated as in (2). A candidate password is checked at one thread.

$$m = m_B \times m_T \quad (2)$$

where m is the number of threads to be allocated to a GPU, m_B is the number of blocks defined in a GPU, and m_T is the number of threads arrayed within a block. m_B and m_T are changeable according to the GPU model and the cracking target.

A kernel is defined using the __global__ declaration specifier, and the number of threads for each call is specified using a new <<<…>>> syntax [26]. Therefore, when a kernel is launched, the dimension of grids and blocks to be executed on the device should be defined. Cracking performance is affected by the execution configuration of a kernel with the optimal use of on-chip memory resources. After many experiments, we define the optimum execution configurations using Geforce 9800 GTX as <<< m_B, m_T >>> = <<<128, 62>>>.

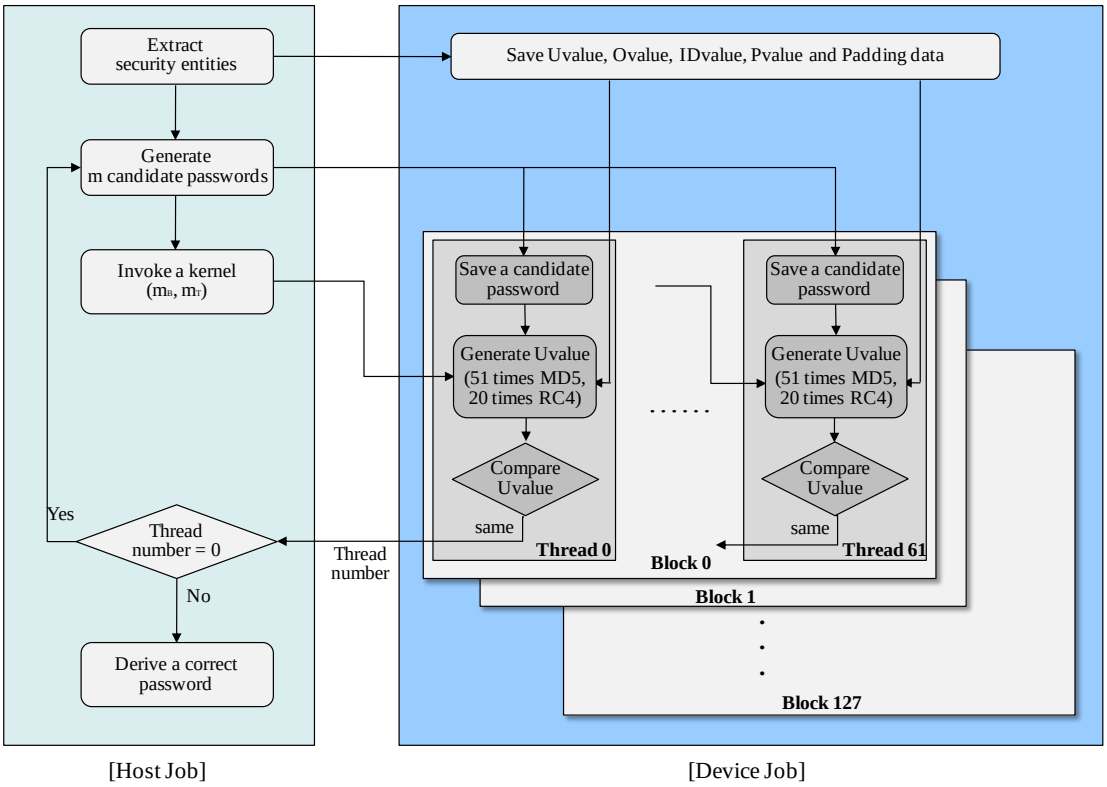

Fig. 3. Password cracking job at a host and a device

The parallel password recovery is actually achieved by the __global__ function, being a kernel on the device. All threads within a GPU operate the same __global__ function except that different candidate passwords are allocated into registers within threads. In other words, the job at thread 0 within block 0 of **Fig. 3** is equal to jobs at all threads in other blocks. Each thread distinguishes candidate passwords by a block index with a grid and a thread index within the block. And, the function of __device__ declaration, which is executed on the device and callable from the device only, is used for MD5 and RC4 cryptographic operation in the __global__ function. If an Uvalue calculated by the __global__ function is not the same as the obtained Uvalue at one of m threads, the host generates a set of new m candidate passwords by a factor of m within N, and the next kernel is invoked in succession.

In order to minimize the access to global memory and to make full use of GPU internal resource, the shared memory is utilized in the ciphering process. The key stream data are stored in the shared memory during RC4 key generation. The RC4 encryption function also places its output into the shared memory. We use 62 threads for a block and 256-byte shared memory for one thread at a __global__ function. Therefore, all the threads of a block use 16KB shared memory. **Table 2** shows the type and size of memory used in the device.

Table 2. Device memory used to recover password

Memory type	Size	Used data
Shared memory	16,140 bytes	Internal data used in key generation and data encryption of RC4 __device__ function
Local memory	192 bytes	Temporary data in __global__ function
Constant memory	116 bytes	Ovalue, Uvalue, IDvalue, Pvalue, and PDF Padding data

If a computed Uvalue matches with an obtained Uvalue at any thread of a device, the device returns its thread number to the host, indicating which block and thread among $m_B \times m_T$ threads is used. The value of thread number is defines as:

$$(BID \times m_T) + TID + 1 \tag{3}$$

where BID is a block index and TID is a thread index. Initial thread number at a host is 0. As soon as the host receives the thread number from the device, it no longer launches a kernel but derives a real password from $32 \times m_B \times m_T$ bytes candidate passwords in the host memory. The remaining passwords of N candidate passwords do not need to be checked anymore. Passwords consist of letters between ASCII code 0x20 and 0x7E. A candidate password is a string less than 32-byte long, and it has additional padding data. The host memory has the same candidate passwords as the device registers. Therefore, the host can extract password letters from the starting point, indicating $(threadnumber-1) \times 32$ until two beginning characters of padding data, 0x28 and 0xBF, sequentially appear. Continuative 0x28 and 0xBF are not letters that can be a part of a password. **Fig. 4** shows how to derive of a real password from candidate passwords at a host. In this figure, we assume that the host generates 7,936 candidate passwords at a time and allocates them to a device. The device returns a thread number with $BID = 95$ and $TID = 32$ at a specific thread. Then, the host derives letters from first character of 5,923th candidate password before padding data. Extracted password is '0x74 0x65 0x73 0x 54 0x37 0x23', that is 'tesT7#'. 5,922th and 5,924th candidate passwords are '0x74 0x65 0x73 0x 54 0x37 0x22' and '0x74 0x65 0x73 0x 54 0x37 0x24', respectively. Search from 5,924th candidates does not progress any more.

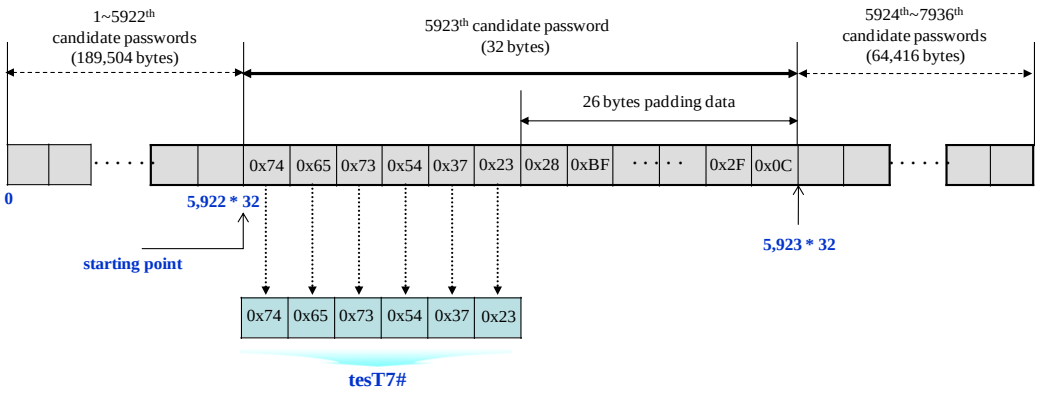

Fig. 4. Derivation of a real password from candidate passwords by a thread number

We also make use of four C1060 devices for GPU acceleration. A C1060 has 240 internal core processers. A host counts the devices and generates $4 \times m$ candidate passwords for four devices at a time unlike using single GPU. Security entities and padding data are copied into each constant memory of four devices, and each device processes m candidate passwords. There is no change at __ global __ and __device__ functions as in using the single GPU. All candidate passwords in the host memory are allocated to registers of four GPUs without duplication as shown in **Fig. 5**.

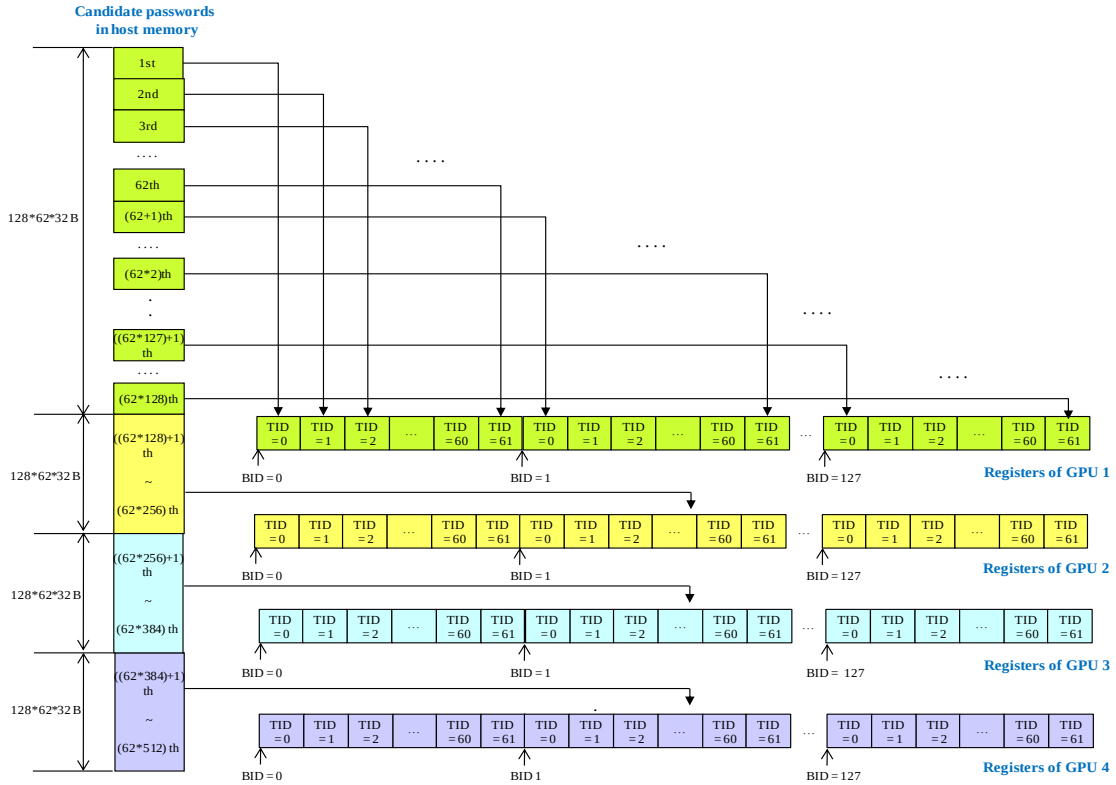

Fig. 5. Allocation of candidate passwords from host memory to four C1060s when 128*4 blocks and 62 threads are defined

4.2 Test and Evaluation

To compare the GPU-accelerated recovery by a brute-force attack with the CPU-based software approach, we measured the cracking time on a single GPU platform and a multi GPUs platform as listed in **Table 3**. A single GPU platform consists of a host machine with Intel Core2 Quad 2.66GHz CPU and 2GB RAM, and Geforce 9800 GTX [27] as a coprocessor for GPU acceleration. A multi GPUs platform has a little more powerful host than a single GPU platform, and uses four Tesla C1060 GPUs [31] instead of 9800 GTX. Two platforms use Linux Fedora 7 OS, and the programming code is written on the CUDA 2.0 software environment. The test by software approach is done on the host machine with Intel Core2 Quad 2.66GHz CPU and 2GB RAM without GPU device. The test passwords are open passwords of Acrobat 8 Standard.

Table 3. Test environment

Platform		Detailed specification
Single GPU	Host	Intel Core2 Quad 2.66GHz 2GB Memory
	Device	Geforce 9800 GTX
Multi GPUs	Host	Intel Xeon E5520 Quad 2.26GHz 4GB Memory
	Device	4 * Tesla C1060
OS		Linux Fedora 7
Programming		CUDA2.0
Test Password		'Document Open Password' of Adobe Acrobat 8 Standard

Table 4 shows the cracking time required for passwords with various lengths and types. Searching begins from one-character passwords. Our test includes passwords consisting of only the last letter, '~', of 95 available characters. So, we can measure the longest cracking time for passwords with the same length, which is the worst case. All candidates are verified by every character combination. Most passwords commonly consist of only lower case letters or combinations of lower case letters and numbers. If upper case letters and special letters are not included in the list of candidate passwords to be checked, search time will be very much reduced compared with using all character combinations. The cracking time is affected by the following conditions.

First, the cracking time increases exponentially as passwords are longer at the use of the same character type. Using a software tool to recover a six-character password of the worst case consisting of '~~~~~~' requires 428 days, while 32 and 7.4 days are required on a single GPU and a multi GPUs platform, respectively. Cracking a seven-character password, '~~~~~~~', demands 8 years using all character types on a single GPU platform. Search time of this case increases extremely compared with the password with length of 6 characters.

Second, the time is affected by the character types used if the lengths of the passwords are the same. Cracking time for '12345' changes according to the character type consisting of candidate passwords. It takes 0.2 seconds for the numeral type and 3.5 minutes for the combination type of numerals and lower case letters together on a single GPU platform.

Third, the recovery time is affected by both the order of character types and the order of characters within the same character type. We check the candidate passwords in the order of lower case letters (from 'a' to 'z'), upper case letters (from 'A' to 'Z'), numbers (from '0' to

‘9’), and special characters (from ‘ ’ to ‘~’). Therefore, some passwords with the same length and character type can be recovered faster than other passwords because of the location of characters. For a simple example, ‘a’ is discovered faster than ‘z’ in 26 one-character candidate passwords of only lower case letters. It takes 20 minutes to discover ‘zzzzzz’ on single GPU, whereas just 1.4 minutes for ‘abcdef’. This is applied likewise to five-character two passwords, ‘~~~~~’ and ‘9f%aM’.

Table 4. Experimental measurements to find out passwords

Password length	Character type	Test passwords	Number of candidates to be checked in the worst case	Search time		
				Software only	Single GPU	Multi GPUs
4	all	~~~~	82,317,120	62 min	5.2 min	1.1 min
		a5F@		55 min	4.6 min	1 min
		abcd		173 sec	13 sec	3 sec
5	all	~~~~~	7,820,126,495	104 hour	8 hour	1.8 hour
		9f%aM		70 hour	5.3 hour	1.2 hour
	numeral	12345	111,110	2.6 sec	0.2 sec	0.05 sec
	lower case and numeral		62,193,780	46 min	3.5 min	0.8 min
6	all	~~~~~~	742,912,017,120	1.1 years	32 day	7.4 day
		h8#D!w		40 day	3 day	0.7 day
		3UK%2r		0.65 year	18 day	4 day
	lower case	abcdef	321,272,406	19 min	1.4 min	0.3 min
		zzzzzz		259 min	20 min	4.5 min
7	all	~~~~~~~	70,576,641,626,495	108 year	8.0 year	1.8 year
	lower case	lovesam	8,353,082,582	18 hour	84 min	19 min
8	all	~~~~~~~~	6,704,780,954,517,120	10,315 years	810 year	175 year
	numeral	12345678	111,111,110	18 min	1.4 min	0.3 min

The results of search time for password length of 7 and 8 using all characters in **Table 4** are the estimate, not the experiment measurement. The time was calculated using the number of all candidate passwords to be searched as the worst case and the cracking performance by test result of length of 4 to 6 since it takes very long to get results.

Most passwords can be recovered in much less time than the worst case time of our measurement. Therefore, cracking time cannot be a factor of cracking performance. The performance should be defined as the number of candidate passwords to be checked a second regardless of the password length, the character types, and the order of characters. In order to crack ‘a5F@’ on a single GPU platform, 7,936 threads repeat 9,220 operations and 73,169,920 of 82,317,120 candidate passwords are checked until a correct password is found. The remaining candidates do not need to be searched anymore, and a correct password is recovered at a speed of 262,000 passwords per second. The password is cracked at a speed of 1,200,000 passwords per second on a multi GPUs platform while 20,600 passwords per second by software approach as shown in **Fig. 6**.

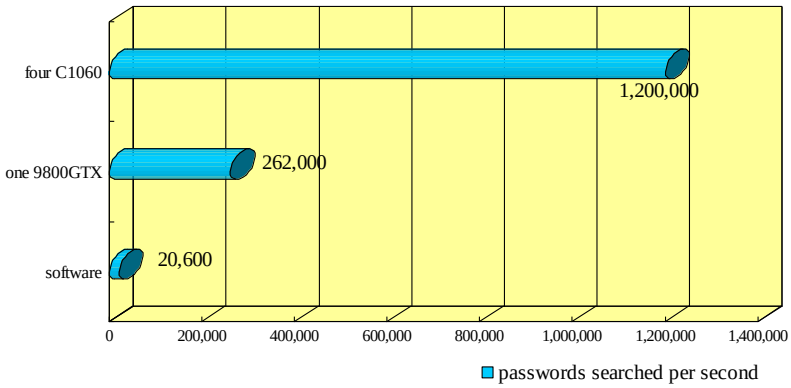

Fig. 6. Password cracking performance

The 9800 GTX has 128 processing cores. It does not mean that we can get a speedup of 128 compared with the software approach since the performance of each core is not much better than a CPU. By many experiments, password cracking by GPU acceleration is at least 13 to 58 times faster than software approach. Performance on a multi GPU platform is more than four times faster than a single GPU platform because a C1060 has more cores than a 9800 GTX. It is expected that the use of more powerful GPU gets better performance.

5. High-Speed Password Recovery Using a Cluster of Computing Nodes

To find out very long passwords, cracking time will increase enormously even it multi GPUs are used. For a high acceleration of password cracking, the job can be distributed over many computers for an additional speedup proportional to the number of available computers with comparable GPUs.

5.1 Password Recovery using GPU Cluster

A GPU cluster system consists of a number of nodes including a control node with user interface, and many work nodes are used to accelerate password search.

Fig. 7 shows the password recovery process between a control node and many work nodes. A control node initiates and terminates the password recovery process. It obtains security entities, selects a set of characters and an expected minimum/maximum password length, and calculates the number of overall candidate passwords. Since all work nodes always inform the control node of their states when they are ready to recover a password, the control node can identify available work nodes and issue password cracking to them. After completing the configuration at the control node, it sends character sets and security entities to all work nodes. And, the control node allocates to all work nodes the starting point of candidate passwords scope to be checked at each work node and the number of candidate passwords for the node.

On receiving the necessary information for password recovery from the control node, the work nodes generate all combinations of candidate passwords beginning from their starting point within the number of candidates that are allocated to them. Password cracking at all work nodes continues until a correct password is found at one of them. Finally, when the control node receives the result about the success of password search from one work node, it orders to terminate the remaining job to the other work nodes. **Table 5** denotes notations used in **Fig. 7**.

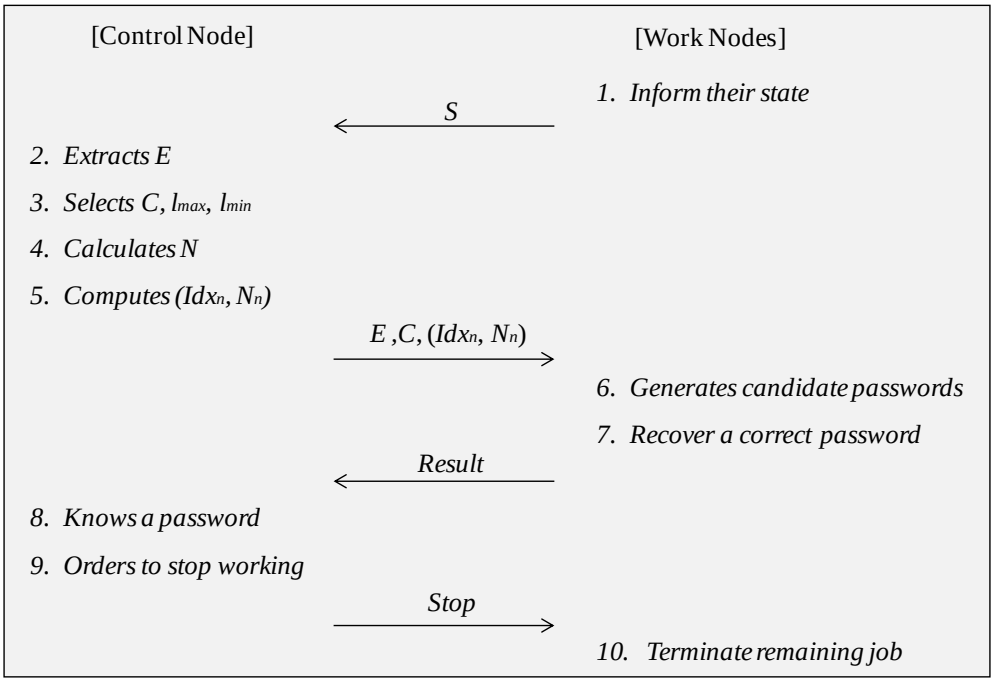

Fig. 7. Password recovery process using GPU Cluster

Table 5. Notation and its meaning for **Fig. 7** and **Fig. 8**

Notation	Meaning
S	State of work node indicating whether it can be used in password recovery
E	Security entities(Ovalue, Uvalue, IDvalue, Pvalue, and padding data)
C	Character set including upper/lower case alphabet letters, arabic numerals, and/or special characters
l_{min}	Expected minimum password length
l_{max}	Expected maximum password length
N	The number of all candidate passwords to be searched. It raises the number to the whole number.
c	The number of available work nodes
m	The number of threads defined at work nodes
D	The number of candidate passwords to be allocated per one work node without considering m
Idx_n	Starting point of candidate passwords to be checked at each work node
N_n	The number of candidate passwords to be allocated to each work node
n	It means n-th work node. If c is 88, n becomes 0, 1,… 87
X	Temporary variable to set multiple of m when D is not multiple of m. It raises the number to the whole number.
Y	The number of candidate passwords to be allocated to a work node considering m

The password recovery result for overlapping candidate passwords at any work node is identical. Therefore, a control node needs to inform the specific starting points to allocate different candidate passwords to all work nodes. And, the control node distributes the same number of candidate passwords to work nodes by considering the number of threads. For instance, 88 work nodes have $(Idx_0, N_0),(Idx_1, N_1),...(Idx_{87}, N_{87})$, respectively, without duplicate candidates among N candidate passwords. If the calculated number of candidate passwords to be allocated per one work node is not a multiple of the number of threads, the node has fewer candidate passwords than other nodes. (Idx_n, N_n) is computed as shown in **Fig. 8**. Notations and meanings are denoted in **Table 5**.

$D = N / c$
If (D is a multiple of m)
$Idx_n = nD$
$N_n = D$
Else
$X = D / m$, $Y = X \times m$
$Idx_n = nY$
$N_n = Y$
If(n is a last work node)
$X = (N - (c-1)Y) / m$, $Y = X \times m$
$N_n = Y$

Fig. 8. Algorithm allocating Idx_n and N_n to each work node

5.2 Performance and Estimated Cost

In this work, the KISTI Picasso system [32] is used as a GPU cluster, which consists of 110 nodes including a control node, an administrative node, a login node, a debugging node, a scheduling node, and work nodes. Every node has an Intel X5450 Quadcore CPU as a host and an Nvidia Quadro FX-5600 GPU [33] as a device. A control node and 88 work nodes from this system are actually used in recovering the passwords. The performance of FX-5600 is a little lower than that of 9800 GTX. We measure the number of candidate passwords to be searched and cracking time.

When 88 work nodes are used, a six-character password, '~~~~~~', is recovered in 10 hours at one of the 88 nodes, while 428 days and 30 days is required by the software approach and the single GPU platform, respectively. To find a seven-character password, 'lovesam', it takes only 30 minutes whereas 563 and 43 hours using software and one GPU. The performance increases linearly with the number of work nodes as shown in **Fig. 9**. When 30 and 88 nodes are involved in recovering a password, 6,700,000 and 19,630,000 passwords per second are searched, respectively. Password recovery using 88 computing nodes is about 1,000 times faster than the software approach.

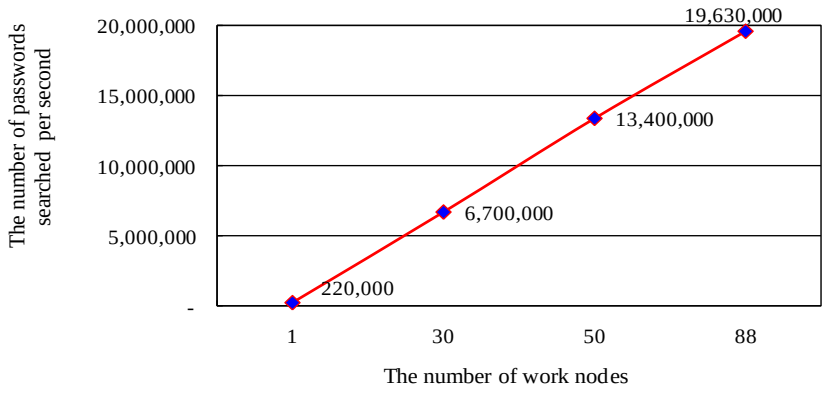

Fig. 9. Password recovery performance according to the number of GPU nodes

To give an idea of the needed resources for the successful cracking of long password in a day, we provide the current estimated cost of the computing nodes in **Table 6**. For passwords using the same character type, the number of computing nodes increases as much as the number of used characters as the password length is increased by 1. For passwords with the same length, the required resources depend greatly on the used character types.

Table 6. The Number of FX-5600 Work Nodes Needed to Recover a Password in a Worst Case in a Day

Password length \ Character type	Digit	Lower	Lower + Digit	Lower + Upper + Digit	Lower + Upper + Digit + Special
8	1	11	152	11,675	352,734
9	1	11 * 26	152 * 36	11,675 * 62	352,734 * 95
10	1	11 * (26^2)	152 * (36^2)	11,675* (62^2)	352,734 * (95^2)
11	6	11 * (26^3)	152 * (36^3)	11,675* (62^3)	352,734 * (95^3)
12	58	11 * (26^4)	152 * (36^4)	11,675* (62^4)	352,734 * (95^4)

The resources required will be significantly reduced compared with FX-5600 if we use the most recent GPU. Since we assume that test passwords consist of only the last characters to measure the maximum time for passwords of the same length, the resource required to find usual passwords will be much less than our estimation cost.

6. Conclusion

Several approaches available to recovering user passwords of encrypted files all have both pros and cons. The use of a password dictionary can recover passwords with simplistic pattern in a short time, but the dictionary does not have all random passwords using various character types. Rainbow table attack allows the password recovery to be executed faster by using the lists of pre-computed password hashes, but this approach is not suitable for password recovery

of common applications with two and more cryptographic algorithms. Brute-force password search tries every character combination as candidate passwords, therefore, it can be a perfect solution for cracking password. However, it is a very time consuming process. This paper focused on a method of accelerating password recovery using GPU. Because GPU is already installed in most computer systems, to use GPU has the advantage of low-cost and high-performance compared with the software approach that uses only the CPU processing.

We proposed a new method of accelerating password recovery of PDF files on GPU. Our experiments on single GPU and multi GPUs verified a significant improvement over the software approach by making a clear job division between a host CPU and a device GPU which affects the search speed. For a drastic speed up of password recovery, we also tested brute-force cracking with a cluster of GPU nodes and proposed a method of taking advantage of all the nodes that are not using the same candidate passwords at every computing node.

Sequential part of the recovery task such as generation of candidate passwords and derivation of the right password runs on the CPU, and compute-intensive part of recovery such as cryptographic computations is performed through data parallelism of GPU. Thus, password recovery using a cluster of 88 GPU nodes is 1,000 times faster than the software approach. Performance is increased linearly in proportion to the number of computing node as well as GPU. Our experimental result also shows that cracking time is affected by the following conditions: (1) the length of the password if the same character type is used; (2) the character types used if the lengths of the passwords are the same; (3) the order of character types; and (4) the order of characters within the same character type. Most passwords will be recovered in much less time than the worst case time of our measurement.

The proposed method may be applicable in applications other than PDF files. Such a trial remains a possibility for future work. On the other hand, GPU may also be used to accelerate to the computation of a rainbow table to support two and more cryptographic algorithms. We will continue to explore the GPU-accelerated password search by using pre-computed table made by GPU acceleration.

References

[1] Debra L. Shinder, Ed Tittel, “Scene of the Cybercrime: Computer Forensics Handbook,” 1st Edition, Syngress Press, Rockland, MA, 2002.

[2] Eoghan Casey, “Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet,” 3rd Edition, Elsevier Academic Press, London, 2011.

[3] S. Marechal, “Advances in Password Cracking,” *Journal in Computer Virology*, vol. 4, no. 1, pp. 73-81, Nov. 2008. Article (CrossRef Link)

[4] A. Narayanan, V. Shmatikov, “Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff,” in *Proc. of 12th ACM Conference on Computer and Communications Security (CCS 2005)*, pp. 364-372, Nov. 7–11, 2005. Article (CrossRef Link)

[5] Martin E. Hellman, “A Cryptanalytic Time-Memory Trade-Off,” *IEEE Transactions on Information Theory*, vol. 26, no. 4, pp. 401-406, July 1980. Article (CrossRef Link)

[6] C. Paar, J. Pelzl, B. Preneel, “Understanding Cryptography: A Textbook for Students and Practitioners,” Springer, 2010.

[7] S.A. Manavski, “CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography,” in *Proc. of IEEE Int. Conference on Signal Processing and Communications (ICSPC 2007),* pp. 65-68, Nov. 24-27, 2007. Article (CrossRef Link)

[8] C. Li, H. Wu, S. Chen, X. Li, D. Guo, “Efficient Implementation for MD5-RC4 Encryption using GPU with CUDA,” in *Proc. of 3rd Int. Conference on Anti-Counterfeiting, Security, and Identification in Communication (ASID 2009)*, pp. 167-170, July 18-20, 2009. Article (CrossRef Link)

[9] W. Zhou, H. Wu, X. Li, D. Guo, "Implementations of Hardware Acceleration for MD4 Family Algorithms based on GPU," in *Proc. of 3rd Int. Conference on Anti-Counterfeiting, Security, and Identification in Communication (ASID 2009)*, pp. 571-574, July 18-20, 2009. Article (CrossRef Link)
[10] R. Szerwinski, T. Guneysu, "Exploiting the Power of GPUs for Asymmetric Cryptography," in *Proc. of 10th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHESS 2008)*, pp. 79-99, August 10-13, 2008. Article (CrossRef Link)
[11] A.D. Biagio, A. Barenghi, G. Agosta, G. Pelosi, "Design of a Parallel AES for Graphics Hardware using the CUDA Framework," in *Proc. of IEEE Int. Symposium on Parallel & Distributed Processing (IPDPS 2009)*, pp.1-8, May 23-29, 2009. Article (CrossRef Link)
[12] J. Yang, J. Goodman, "Symmetric Key Cryptography on Modern Graphics Hardware," in *Proc. of Advances in Cryptology (ASIACRYPT 2007)*, pp. 249-264, Dec. 2-6, 2007. Article (CrossRef Link)
[13] Elcomsoft. http://www.elcomsoft.com/distributed_password_recovery.html
[14] K. Jang, S. Han, S. Han, S. Moon, K. Park, "SSLShader: Cheap SSL Acceleration with Commodity Processors," in *Proc. of 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*, pp. 1-1, Mar. 30-Apr. 1, 2011. Article (CrossRef Link)
[15] S. Han, K. Jang, K. Park, S. Moon, "PacketShader: A GPU-Accelerated Software Router," in *Proc. of ACM SIGCOMM 2010 Conference on SIGGCOM*, pp. 195-206, Aug. 30-Sep. 3, 2010. Article (CrossRef Link)
[16] V. Garcia, E. Debreuve, F. Nielsen, M. Barlaud, "K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching," in *Proc. of 17th IEEE Int. Conference on Image Processing (ICIP 2010)*, pp. 3757-3760, Sep. 26-29, 2010. Article (CrossRef Link)
[17] Y. Zhuge, Y. Cao, R.W. Miller, "GPU Accelerated Fuzzy Connected Image Segmentation by using CUDA," in *Proc. of Annual IEEE Int. Conference on Engineering in Medicine and Biology Society (EMBS 2009)* , pp. 6241-6344, Sep. 3-6, 2009. Article (CrossRef Link)
[18] Electronic Frontier Foundation. http://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker
[19] Electronic Frontier Foundation, "Cracking Des: Secrets of Encryption Research, Wiretap Politics & Chip Design," 1st Edition, O'Reilly Media, 1998.
[20] Copacobana. http://www.copacobana.org
[21] Sciengines. http://www.sciengines.com
[22] T. Guneysu, T. Kasper, M. Novotny, C. Paar, A. Rupp, "Cryptanalysis with COPACOBANA," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1498-1513, Nov. 2008. Article (CrossRef Link)
[23] J.A. Kahle, M.N. Day, H.P. Hofstee, C.R. Johns, T.R. Maeurer, D. Shippy, "Introduction to the Cell Multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4.5, pp. 589-604, July 2005. Article (CrossRef Link)
[24] N. Breeze, "Crackstation : Optimized Cryptography on the Playstation3," 2007.
[25] Tableau TACC 1441, http://www.tableau.com
[26] Nvidia Corp., "NVIDIA CUDA Compute Unified Device Architecture Programming Guide," Version 2.0, 2008.
[27] Geforce 9800 GTX, http://www.nvidia.com/object/product_geforce_9800m_gtx_us.html
[28] Tesla S1070, http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_S1070_US_Jun08_NV_LR_Final.pdf
[29] R. Basu, S. Ganguly, S. Maitra, G. Paul, "A Complete Characterization of the Evolution of RC4 Pseudo Random Generation Algorithm," *Journal of Mathematical Cryptology*, vol. 2, no. 3, pp. 257-289, Oct. 2008. Article (CrossRef Link)
[30] R.L. Rivest, "The MD5 Message-Digest Algorithm," Internet RFC 1321, April, 1992.
[31] Tesla C1060, http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C1060_US_Jan10_lores_r1.pdf
[32] Picasso system. http://www.ksc.re.kr/eng
[33] Quadro FX-5600. http://www.nvidia.com/object/quadro_fx_5600_4600.html

Keonwoo Kim received his B.S. and M.S. degrees in electronic engineering in 1999 and 2001, respectively, from Kyungpook National University in Korea. He is currently a senior member of engineering staff at Cryptography Research team in ETRI, Korea. His main research interests are security in mobile communication, cryptography, digital forensics, and data visualization.

Sang-Su Lee received his B.S. and M.S. degrees in electronic engineering in 1999 and 2001, respectively, from Kyungpook National University, Korea. He has been a staff of engineering of Network Security Dep. in ETRI, Korea since 2001. His research interests include digital security, optical security, and digital forensics.

Dowon Hong received his B.S., M.S. and Ph.D. degrees in mathematics from Korea University, Seoul, Korea on 1994, 1996, and 2000. He is currently a principal member of engineering staff and the team leader of Cryptography Research team at the Electronics and Telecommunication Research Institute, Korea where his research interests are broadly in the area of applied cryptography, networks security, and digital forensics.

Jae-Cheol Ryou is a professor at Department of Computer Engineering in Chungnam National University in Korea. He is also the director of the Internet Intrusion Response Technology Research Center (IIRTRC), Chungnam National University, Korea. He received the B.S. degree in Industrial Engineering from Hanyang University in 1985, the M.S. degree in Computer Science from Iowa State University in 1988, and the Ph.D. degree in Electrical Engineering and Computer Science from Northwestern University in 1990. His research interests are Internet Security and Electronic Payment Systems.