A Homogeneous Parallel Brute Force Cracking Algorithm on the GPU *

Anh-Duy Vu, Jea-Il Han**, Hong-An Nguyen, Young-Man Kim, Eun-Jin Im
School of Computer Science, Kookmin University
Email: {anhduyvu, hongan, ymkim, jhan, ejim}@kookmin.ac.kr

Abstract – From the early days of computing, passwords have been considered as the essential authentication method to protect accesses to computer systems and users. Due to their importance, sensitiveness and confidentiality, many cryptography mechanisms have been utilized to secure password storage. Among them, cryptography hash methods are the most popular solutions. A cryptography hash function converts plaintext passwords to unreadable message digests which frustrates attackers from exploiting system failures and stealing stored passwords. On the other hand, it is possible to get the plaintext passwords from digests. We examined brute force attack to get the original passwords from the hashed ones and studied some existing GPU-based brute force cracking tools. These applications implement a hybrid algorithm that generates available passwords on CPU side and hashes them in parallel on GPU side. In this paper, we propose a new homogeneous parallel brute force cracking algorithm that performs all the works on GPU side. In our experiments, we successfully cracked many kinds of passwords. For example, with 6-digit passwords, it took about 0.23 ms for initialization, 1.97 ms for combination generation, and 52.81 ms for brute-force. So we need less than 1 second to crack passwords of this kind.

Keywords: Brute-force cracking, hashed password, General Purpose Graphic Processing Unit (GP GPU), GPU computing, CUDA.

I. INTRODUCTION

Nowadays, passwords are used for user authentication in almost all applications including local machine accounts, domain accounts, email accounts, and etc. These passwords could be stored in various security degrees such as plaintext, applying simple hash algorithms (SHA1, MD5, and etc,) and applying salted hash algorithms with multiple iterations as in UNIX implementation. In theory, all hash algorithms could be cracked with long enough time and currently there are two main approaches to crack a hash algorithm.

- The first is called collision attack which directly attacks weak points of the algorithm by trying to find two arbitrary inputs that will produce the same hash value. In order to be successful, attacker must be in control of the inputs to the hash function. Thus, it is used mostly to attack digital signatures.
- The second is pre-image attack, and brute force attack is a representative of this approach. In this way, the attacker already has a digest (called target) and tries to find out the plaintext by hashing all available messages. The plaintext is one that has the digest exactly like the target. The cracking time depends on the length of messages and the size of character set composing the messages. So this approach is suitable only for cracking short passwords (usually from 6~12 characters.)

The General Purpose Graphic Processing Unit (GP-GPU) executes code on the massive parallel stream processing hardware located in the modern video cards, with performance previously considered that of supercomputer. The appearance and contribution of GP-GPU also affect the possibility of implement a good brute force cracking hash password application. In this paper, we examine some existing brute force cracking solutions that utilize GP-GPU to improve the performance, identify their weakness and propose a new algorithm.

The rest of this paper is organized as follows. In section 2, we discuss the properties which make a password strong and the relationship between the password strength and cracking time. Section 3 presents the related research and problems. In section 4, we explain the novel homogenous parallel cracking algorithm and its implementation in CUDA framework to crack SHA1-based passwords. Finally, our conclusions and future works are presented in section 5.

II. PASSWORD STRENGTH EVALUATION

Password strength can be measured in the amount of cracking time. There are several factors that can make a password more complex and harder to be cracked. Some factors are mentioned in [1] such as: (1) including punctuation marks and/or numbers, (2) mixing capital and lowercase letters. It is observed that a good password must be hard to guess, have an acceptable length and include characters chosen from a large character pool. The Figure 1 illustrates the password strength evaluation of Gmail.

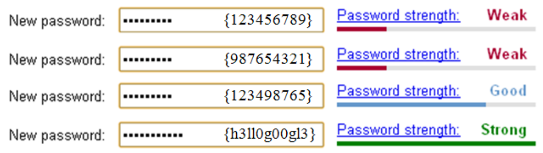

New password: ••••••••• {123456789} Password strength: Weak
New password: ••••••••• {987654321} Password strength: Weak
New password: ••••••••• {123498765} Password strength: Good
New password: ••••••••••• {h3ll0g00gl3} Password strength: Strong

Figure 1. Gmail password strength evaluation [3]

The complexity of a password is measured in entropy which is computed by the length of the password and size of character pool. The entropy of a password is typically stated in terms of bits. For example, a known password has zero bit

* This work was supported by research program 2011 of Kookmin University in Korea
** Corresponding author

of entropy, while a password with 1 bit of entropy would be guessed on the first attempt 50% of the time. A password with n bits of entropy would be as difficult to guess as an n bit random quantity. In general, a password with n bits of entropy can be found in 2^n attempts [2] and obviously it takes more time to crack. The below formulae show how to calculate entropy of a password

$$Entropy\_per\_char = \log_2(n) \quad (1)$$

$$Password\_entropy = l * Entropy\_per\_char \quad (2)$$

where n is the pool size and l is password length

Table I denotes the relationship between character pools and entropy per character for each pool.

TABLE I. ENTROPY PER CHARACTER FOR SOME CHARACTER POOLS

Character Pool	Pool size	Entropy Per Character
Digits	10 (0-9)	3.32 bits
Lower-case letters	26 (a-z)	4.7 bits
Lower-case letter and digits	36(a-z, 0-9)	5.17 bits
Case sensitive letters	52 (A-Z, a-z)	5.7 bits
Case sensitive letters and digits	62(A-Z, a-z, 0-9)	5.95 bits
All standard keyboard characters	94	6.55 bits

III. RELATED WORK

We have noticed that several studies examined the feasibility of parallel hash crackers but there was no published paper in this field. Some GPU-based Brute Force applications are presented in Table II.

TABLE II. SOME GPU-BASED BRUTE FORCE CRACKING TOOLS

Name	Hashes	Operating system	License
Elcomsoft [4]	MD5, NTLM, DCC	Win 32	MD5 is free. Complete package is commercial
GPU md5 Crack [5]	MD5	Linux 32 Win32	LGPL v3
Multihash CUDA Brute Forcer [6]	MD5, NTLM	Linux 32/ 64 Win 32	Unknown
Extreme GPU BruteForcer [7]	MD5, NTLM	Win 32	Shareware
Distributed Hash Cracker [8]	MD5, NTLM	Linux 32/ 64	BSD
IGHASHGPU [9]	MD5, NTLM, DCC, ORACLE 11g	Win 32	Free for non-commercial use
BarsWF bruteforce [10]	MD5	Win 32/ 64	Unknown
Rainbow Crack [11]	MD5, NTLM, ORACLE 11g	Win 32	Unknown

Among these tools:

- Only [4] and [8] support distributed GPU-based hash cracking
- Only [5] and [6] support multiple operating system platforms
- [4], [7], [9], [10], and [11] only run on Window operating system and are close source products which cannot be studied at source code level
- Only [5] is the open source tool that can be studied at code level

As reported in [12], some of these tools were executed to crack MD5 and the comparison of their cracking time is presented in Figure 2. The experiments were deployed on Windows Server 2008 x64 host with ASUS ENGTX 295 graphic card. This card has NVIDIA GeForce GTX 295 chip including 480 processor cores with a core clock speed of 1.242 GHz.

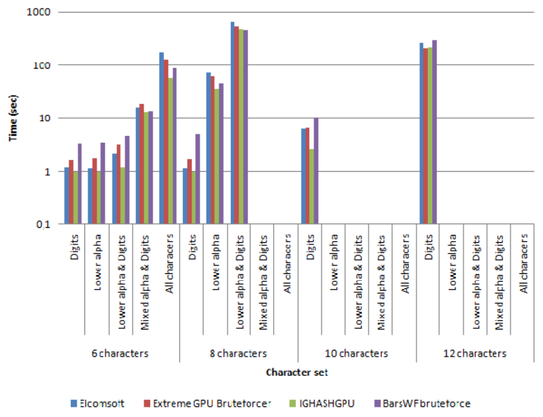


Figure 2. The time consumed for cracking MD5 hashes with different password lengths and character sets

The analysis of the source code of [5] shows that there are two main steps utilized in the hybrid brute force cracking algorithm:

- Step 1: Generate all candidates at host (CPU) side
- Step 2: In parallel, hash all the candidates then compare each result with the target until the plain text password is found.

It is observed that this approach can achieve better performance if the first step is also performed in GPU.

IV. HOMOGENUOUS PARALLEL CRACKING ALGORITHM AND IMPLEMENTATION

For a character pool of size n and passwords of length l, there are n^l candidates. Generating and storing such amount of strings at one time on GPU is not a good approach. Besides, in the loop for generating candidates, the results are put into an array in ascending order. So if we try each element of this array from the first to the last, it takes longer to crack passwords composed by characters from the end of the pool. These are the reasons why we do not generate all the available strings directly but only all multi-combinations of the pool. The candidates are the permutations of the combinations. For example, when the pool has 3 characters {1, 2, 3}, all 3-character candidates of this pool are the permutations of its multi-combinations as denoted in Table III.

TABLE III. COMBINATIONS AND THEIR PERMUTATIONS

Combination	**Permutations**
111	111
112	112, 121, 211
113	113, 131, 311
122	122, 212, 221
123	123, 132, 213, 231, 321, 312
133	133, 313, 331
222	222
223	223, 232, 322
233	233, 323, 322
333	333

The number of multi-combination is calculated by the following formula:

$$m = \frac{n*(n+1)*...*(n+l-1)}{l!} \qquad (3)$$

where m is number of multi-combinations, n is size of character pool, and l is password length

A. *Homogenous parallel cracking algorithm*

It is assumed that a password is composed of 3 characters ($p_0p_1p_2$) from character pool *{1, 2, 3}*. Traditionally, a set of 3 iterations is utilized to generate all candidates and put them into a buffer. The number of iterations can be reduced by means of parallelism. For generating multi-combinations on GPU, the first character p_0 and the second one p_1 are handled simultaneously by blocks and threads, and the remaining character is still handled by an iteration. All the combinations are stored in a single buffer and it is shared to all threads in blocks. To determine the initial position for each thread to write down their results into the buffer, we

- Utilize formula (3) to count the number of multi-combinations generated by each thread and keep these values in *com_thread* array
- Perform prefix sum algorithm for *com_thread* array

This process is presented in Figure 3.

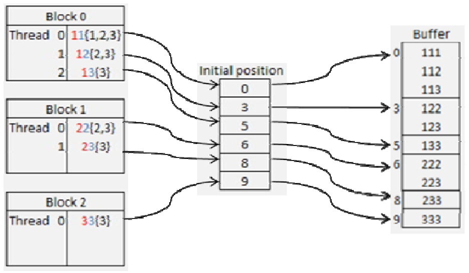


Figure 3. Generating candidates on GPU side

For generating permutation, the most common methods are based on recursive functions. However, GPU environment does not support recursive methods, so we recommend the algorithm of Donald E. Knuth [14] which utilizes iterations to generate all permutations in lexicographic order.

Algorithm 1 describes entire process for cracking 3-character passwords. The algorithm can be extended to crack n-character passwords by adding more iterations in *generate combination* section of phase 2.

```
Phase 1: Initialize
 1: count number of all multi-combinations
 2: allocate buffer space
 3: count number of combinations of each threads (com_thread[])
 4: perform prefix sum on com_thread[] to determine init_pos[]
Phase 2: Generate multi-combinations
 1: for each thread t in block b in parallel do
 2:    for i = 0 → b do
 3:       idx = idx + char_pool_size – i
 4:    end for
 5:    start_pos = init_pos[idx + t]* pwdLen // in this case pwdLen is 3
 6:    for p2 = t → char_pool_size do // generate combinations
 7:       buffer[start_pos] = char_pool [b]
 8:       buffer[start_pos + 1] = char_pool[t]
 9:       buffer[start_pos + 2] = char_pool[p2]
10:       start_pos = start_pos + pwdLen
11:    end for
12: end for
Phase 3: Brute Force
 1: for each thread t in block b in parallel do
 2:    get a candidate c out of buffer
 3:    for each permutation e of c do
 4:       r = hash(e)
 5:       if r equals target do
 6:          finish
 7:       end if
 8:    end for
 9: end for
```

Algorithm 1. Brute Force cracking hashed passwords

B. *Implementation and results*

To evaluate the new algorithm, we implemented it in CUDA framework 3.0 under Fedora 10 x64 to crack SHA1. We also modified the SHA1 implementation of Packetizer Community [13] to be able to run on CUDA framework. In the experiments, we divided passwords into 5 various categories:

- Category 1 contains only numeric characters (0-9).
- Category 2 contains only alphabet characters in lower case (a-z).
- Category 3 is a combination of numeric characters and alphabet characters in lower case (0-9, a-z).
- Category 4 is a combination of case sensitive alphabet characters (a-z, A-Z).
- Category 5 is a combination of numeric and case sensitive characters (0-9, a-z, A-Z).

For each category, we tried to crack several different passwords, and measured the cracking time. Figure 4 shows the comparison of time consumed for cracking SHA1 of 6-

character passwords on Tesla C1060 (including 240 cores and a core clock speed of 1.3 GHz) with those on Tesla C2050 (including 480 cores and a core clock speed of 1.15 GHz.) The results show that when executed on Tesla C2050, the algorithm offers a better performance. For digit-passwords, it takes less than 1 second to crack, so this kind of password should not be used for important accounts. The cracking time increases greatly and unpredictably when the character pool becomes larger.

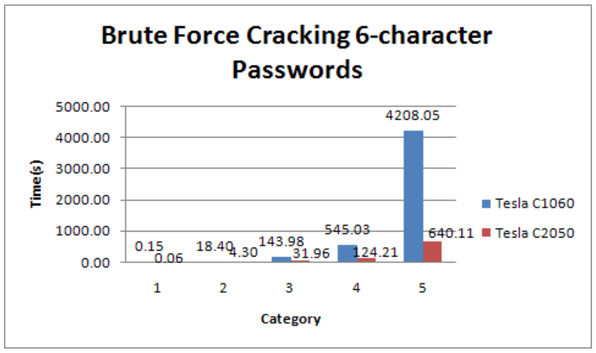
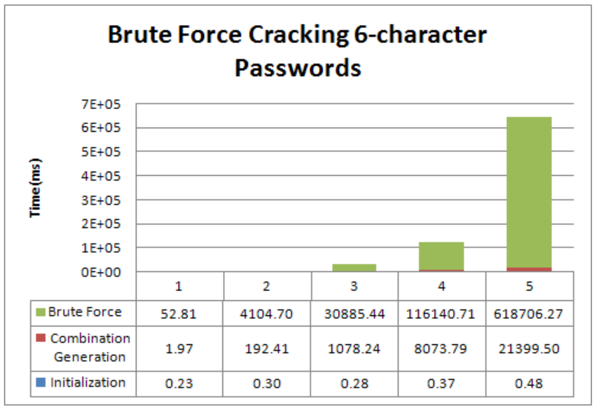


Figure 4. Brute Force cracking 6-character Passwords on Tesla C1060 and C2050

In the experiments, we also measured computing time of each phase for categories and showed in Figure 5.



	1	2	3	4	5
🟩 Brute Force	52.81	4104.70	30885.44	116140.71	618706.27
🟥 Combination Generation	1.97	192.41	1078.24	8073.79	21399.50
🟦 Initialization	0.23	0.30	0.28	0.37	0.48

Figure 5. Figure 1 Phases comparison on Tesla C2050

C. Limitation

In formula (3), if the size of character pool is 94 (all standard keyboard characters) and password length is 6, there are about 10^9 combinations. So we need about 6 GB of memory to store these combinations which is not practical. To solve this issue, we intend to optimize our current algorithm so that it can divide the character pool into several smaller subsets and work with each of them.

V. Conclusion

Passwords have been considered the fundamental authentication method to protect computer systems and users in various applications such as in email, bank, network... They can be stored as plaintexts, simple hash results of MD5, SHA1, and DCC … or salted hash results. Due to some system vulnerability, an attacker can steal the stored password and try to crack the passwords. With the support of GPU, this work has become easier and faster.

In this paper, we studied some existing GPU-based brute force cracking tools, pointed out their weakness and proposed a new algorithm. The experimental result shows that it takes less than 1 second to crack 6-digit passwords. So as users, it must be safer to choose passwords longer and more complex by combining numeric, case sensitive and special characters. And as administrators, we must use salted hash algorithm to protect our password storage.

Although the new solution supports to generate candidates on GPU side, it has problems with memory usage while cracking passwords composed from large character pool. We are still working to improve the algorithm not only to solve the issue but also to reduce the time consumed and support cracking salted hashed passwords.

Reference

[1] Password advices from Google, May 2011. URL https://www.google.com/accounts/PasswordHelp
[2] Random password strength. URL http://www.redkestrel.co.uk/Articles/RandomPasswordStrength.html
[3] Google Accounts change password. URL https://www.google.com/accounts/EditPasswd?hl=en-GB
[4] Elcomsoft Proactive Software Company. URL http://elcomsoft.com/
[5] GPU md5 crack. URL http://bvernoux.free.fr/md5/index.php
[6] PenTestIT. URL http://www.pentestit.com/tag/multihash-brute-forcer/
[7] InsidePro – Password recovery software. URL http://www.insidepro.com/
[8] "Distributed Hash Cracker" [Online document], available at http://www.cs.rpi.edu/~zonena/papers/cracker.pdf
[9] IGHASHGPU. URL http://www.golubev.com/blog/?tag=ighashgpu
[10] BarsWF bruteforce. URL http://www.darknet.org.uk/2008/12/the-worlds-fastest-md5-cracker-barswf/
[11] RainbowCrack project. URL http://project-rainbowcrack.com/
[12] GPU-based password cracking. URL http://staff.science.uva.nl/~delaat/sne-2009-2010/p34/report.pdf
[13] SHA1 of Packetizer Community. URL http://www.packetizer.com/security/sha1/
[14] Donald E. Knuth. The art of computer programming volume 4A