Dictionary Attacks for the Franko-Arabic Password using GPUs

Mohamed A. Khalil, Naglaa M. Reda, H. M. B. Ibrahim

Computer Science Division, Department of Mathematics, Faculty of Science, Ain Shams University, Cairo 11566, Egypt

Mohamed.ali@mof.gov.eg, naglaa_reda@sci.asu.edu.eg, h.m.bahig@gmail.com

Abstract—A password is the most important technique for a user authentication in application software. It is commonly secured by hashing it. In this case, we call the hash value a hashed password. After that, the hashed password is saved in a password table. When a user logs in his account, for example, this password is hashed again and compared with the saved hashed password in the password table to authorize user access. Since graphics processing unit (GPU) hardware has recently advanced, we are interested to show the significant speedup of GPUs over traditional central processing units (CPU's) in cracking passwords generated from the Franko-Arabic language and hashed using SHA-3. We present the possibility of cracking the passwords constructed from Franko-Arabic language and hashed by SHA3. We have been able to achieve high performance of cracking passwords using GPUs. As a result, our GPU implementation is faster than highly optimized processors CPU. With this modern development and increased performance, "complex" passwords with up to 10 characters becoming possible to crack. To the best of our knowledge, this is the first paper that discusses the cryptanalysis (or cracking) passwords constructed from the Franko-Arabic language.

Index Terms— Hash function, SHA-3, Password cracking, CUDA, GPU, Dictionary attack, Franko-Arabic language

1 Introduction

Hash functions [9] are very important aspect to achieve some security issues such as message and entity authentications and digital signatures. Users use passwords to authenticate themselves to computer systems, while attackers try to get those passwords back.

A hash function H is a function that maps arbitrary binary strings (input data) into binary strings of fixed length (output data).

$$H: \{0,1\}^* \rightarrow \{0,1\}^n, \quad m \mapsto H(m)$$

The hash function H should be efficiently applied to input of different sizes.

The length of the output is less than that of the input, which can be considered the reason why a hash function is called a compression function. On the other hand, a compression function cannot be considered a hash function.

A hash function H is said to be secure if the following conditions are satisfied.

1- Pre-Image Resistance (One-way property)

Given $h \in \{0, 1\}^n$, it is computationally infeasible to find $m \in \{0, 1\}^*$ such that H(m)= h. This property measures how difficult it is to put a message m which hashes in a known digest h. Therefore, H must be a one-way function, i.e., it is computationally infeasible to reverse the output of the hash function. This property ensures that an attacker cannot return, for example, a password if it is hashed.

2- Second Pre-Image Resistance (weak collision resistant)

Given $m \in \{0, 1\}^*$, it is computationally infeasible to find $m' \in \{0, 1\}^*$, $m \neq m'$ such that H(m) = H(m'). This property measures how difficult it is to devise of a hashes message into a known digest and its message.

This property prevents an attacker, who knowing the message m and its hash H(m), to replace the message m with a different message m', i.e., legitimate message instead of the original message m.

3- Collision Resistance (strong collision resistant)

It is computationally infeasible to find two distinct messages m, $m' \in \{0, 1\}^*$, such that H(m') = H(m).

This property is also called the collision free hash function. Since the hash function is a fixed length compression function, it is impossible for a hash function not to have collisions. In addition, if the hash function is collision resistant, then it is second pre-image resistant.

When a hash function verifies the above three conditions, we can say that a hash function is secure.

During the last 30 years, there are many proposed hash functions, such as MD5, SHA-0, and SHA-1. But after successful attacks on MD5, SHA-0, and SHA-1 [2][3][4] , National Institute of Standards and Technology (NIST) decided not to use any one of them and to continue using SHA-2, since SHA-2 is still secure. But NIST purposed SHA-3 to be ready substituted for SHA-2 in computer system if necessary. So, we decided to study the dictionary attack for Franko-Arabic passwords hashed by secure hash function SHA-3.

To the best of our knowledge, there is no article discuss the cryptanalysis (or cracking) passwords constructed from the Franko-Arabic language.

This paper is organized as follows. In Section 2, we describe the Franko-Arabic language and why we choose it as the characters set of the constructed passwords. In Section 3, we give an overview of graphics processor units and compute unified device architecture since we will use them in cracking passwords. In Section 4, we describe a dictionary attack. In Section 5, we present our experiment to crack passwords. The conclusion is presented in Section 6.

2 The Franko-Arabic Language

The Franco-Arabic language is a newly popular writing style in the Arab world which the Latin alphabet and numbers are used to replace certain Arabic letters in order to write Arabic words phonetically. It isn't truly Arabic, nor is it truly English. It's common for characters in the Franco-Arabic language to use the Latin character that best resembles the sound of the Arabic letter that would otherwise be used.

Due to some reasons such as rising popularity of some software and technologies, such as chatting, SMS, social media and mobile phones, the Franko-Arabic language is used rapidly in Arabic countries.

The set of characters of passwords is presented in Table 1. Table 2 shows some examples of Franko-Arabic words.

Table 1 Franko-Arabic characters

Character in Arabic	أ	ب	ت	ث	ج
Character in Franko-Arabic	a	b	t	th	j
Character in Arabic	ح	خ	د	ذ	ر
Character in Franko-Arabic	7	kh	d	d‘	R
Character in Arabic	ز	س	ش	ص	ض
Character in Franko-Arabic	z	s	$, ch , sh	9 , S	9‘ , D
Character in Arabic	ط	ظ	ع	غ	ف
Character in Franko-Arabic	6	6‘ , Z	3	3‘ , gh	F
Character in Arabic	ق	ك	ل	م	ن
Character in Franko-Arabic	8 , q	k	l	m	n
Character in Arabic	هـ	و	ي	ء	
Character in Franko-Arabic	h	W , o	i , Y	2	

Table 2 Examples of Franko-Arabic password with different sizes

Word(s) in Franko-Arabic	Word(s) in Arab
t3raf	تعرف
7bibi	حبيبى
8mr	قمر
al5yr	الخير
mats2lsh	ماتسئلش
th3lb	ثعلب
5r6om	خرطوم

3 GPUs and CUDA

Due to a high computational power of graphics processor units (GPUs), they are considered one of the most promising technologies in the parallel computing field. GPUs have improved the performance of many problems in different fields of research, such as cryptanalysis of some cryptosystems [8]. Thousands of cores are in a single GPU. GPUs are considered a heterogeneous model that works in conjunction with the central processing unit, as seen in Fig. 1.

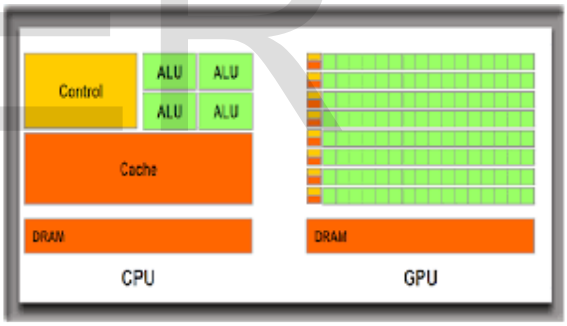

Fig. 1. GPU vs. CPU.

NVIDIA [1] invented Compute Unified Device Architecture (CUDA), a general-purpose parallel programming model. It makes use of the parallel compute engine built into NVIDIA's GPUs to accelerate computational algorithms beyond the capabilities of the CPU. CUDA creates parallel programs by extending standard languages such as the C language. The CPU initiates CUDA programs and sends instructions to the GPU, which acts as a coprocessor. Because the GPU and CPU have distinct memory architectures, it is necessary to move data to the GPU memory prior to the GPU performing any data manipulations and vice versa. Kernel functions are those that run on the GPU. At runtime, the GPU is in charge of parallelizing the execution of kernel functions by utilizing threads as "execution resources". Threads are organized in blocks and communicate via shared memory. CUDA performs thread synchronization via "barriers". Threads are executed in CUDA in warps; all threads within a warp execute the same instruction

concurrently. Branching in conditional and iteration statements are the most common code constructs that can result in warp divergence.

4 Dictionary Attacks

There are different models to crack passwords. One of them is a dictionary attack. In this model of attack, the attacker uses pre-defined lists of words/passwords or a simple combination of several passwords. Examples of a dictionary attack model are [6] [7]. There is no dictionary attack on Franko-Arabic passwords. Thus, we are interested to recover the passwords using dictionary attack.
We store, in a database, for each possible word/password in a given dictionary the corresponding hash code. If an attacker intercepts a hash code for a password, then he looks it up in the database, and finds the matching password.
Let L be the set of words in a given dictionary. In general, the probability of a successful attack is

$$|L|/N$$

Where $|L|$ denotes the number of words/passwords in the given dictionary, and N denotes the number of possible words of length less than or equal to N.

5 Experimental Implementation

This section presents the implementation details of our experiment to crak passwords. It consists of three subsections. In Subsection 5.1, we describe the setting of the experimental studies including data set, hardware, and software used. In Subsection 5.2, we present our implementation to crack passwords in some details. Subsection 5.3 contains some examples.

5.1 PLATFORM OF EXPERIMENT

The implementation was done using the C++ language and CUDA language (V.9.0). We also used MSSQL to store the dictionary in a database. The specification of GPUs cards used in our implementation are presented in Table 3. The computer ran Microsoft Windows 10 operating system and with a speed of 2.6 GHz and a memory of 8 GB.

Table 3 Specification of two GPUs cards

Item	GPU-1	GPU-2
Device name:	GeForce GTX 1060 6GB	GeForce GTX 770 2GB
Memory Clock Rate (KHz)	4004000	3600000
Memory Bus Width (bits):	192	256
Peak Memory Bandwidth (GB/s):	192.192000	230.400000
Global memory:	6144mb	2048mb
Shared memory:	48kb	48kb
Constant memory:	64kb	64kb
Block registers:	65536	65536
Warp size:	32	32
Threads per block:	1024	1024
Max block dimensions	[64 ,1024 ,1024]	[64 ,1024 ,1024]
Max grid dimensions	[65535 ,2147483647]	,2147483647] [65535

5.2 IMPLENTATION DETAILS

We have generated a dictionary contains 22472030 Franko-Arabic passwords. Then the following steps are done to crack passwords.

a) We created a sequential application using the C++ language that calculates the hash value for passwords in the given dictionary. It takes 2 days, 13 hours, and 51 minutes.

b) We created an application using the CUDA language that calculates the hash values for passwords in the given dictionary. Then we stored passwords and their corresponding hash value in MSSQL server databases. This is done by distributing millions of passwords on two GPUs cards. We reserved 44 blocks; each block has 1024 threads for each card. Thus, the number of passwords sent for each GPU card is equal to the number of blocks multiplied by the number of threads which is 44 * 1024 = 45056 passwords. Fig. 2 shows the application's workflow. After finishing calculating hash values, we store in a database the passwords and the corresponding hash values. This process took about 22 hours and 42 minutes. Clearly, generating hash values all for passwords in a dictionary using GPUs is faster than generating them without GPUs.

c) Now, given a hash value, it is easy in less than 1 second to return the corresponding password or return "not exist".

d) Increasing the size of dictionary means increasing the chance to return the correct password.

e) We have checked if there is a collision in the database. We have found that, there is no collision in the calculated hash values.

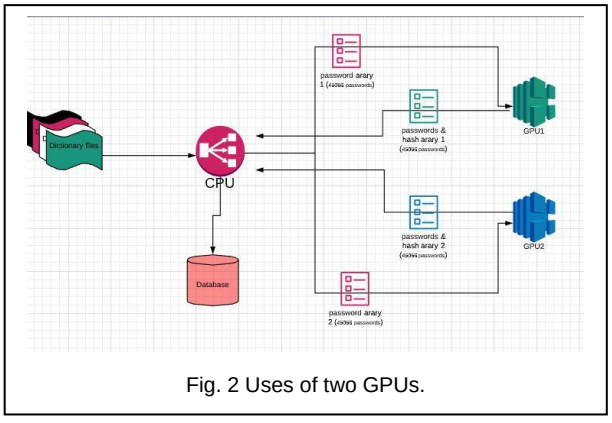

Fig. 2 Uses of two GPUs.

5.3 EXAMPLES

The following are examples of hash values for some Franko-

Arabic words presented in Table 2 Examples of Franko-Arabic passwords.

The hash value of "t3raf" is

c359c83770d2d015e6f0ba9f4eeecdac324314f4d9977606a58bd90a6fc0ac31c30f7e6ac8dea19aeebb0340144e563aabdf7e12447ff5b88f312f0866c878d8.

The hash value of "al5yr" is

d098b51869ef7364a291f39c09bb6a955ed93cc8e3e921c47c392a115cb67b81c86e8ddce31d0bf9a9f58b7cd57182d2e01201685d920f2f0609a7b2e314b3d9.

The hash value of "5r6om" is

c10f9c136a89e243b90627eb0f55d43f091324927fdd2e83f4c9da7e275bdd71fee0a451abf9ff07fae7bcd477670d2e718688e33568bedc3ed470b222fa2917.

4 Conclusion

This paper has studied the possibility to crack passwords constructed from Franko-Arabic language and hashed by SHA-3. Authors have generated a dictionary of 22472030 Franko-Arabic passwords and calculated their hash values. Expermints shows that if the password is in the dictionary, then the presented program can easily crack the hashed password. This was achieved by distributing the passwords on two GPUs cards, and the use of multithreding with CUDA.

References

[1] NVIDIA, "Nvidia cuda c programming guide, cuda toolkit documentation," In NVIDIA CUDA C Programming Guide, CUDA Toolkit documentation, 2017.

[2] C. R. Biham E., "Near-Collisions of SHA-0," in *In: Franklin M. (eds) Advances in Cryptology – CRYPTO 2004. CRYPTO 2004. Lecture Notes in Computer Science, vol 3152. Springer*, Berlin, Heidelberg, 2004.

[3] D. Chad R, "Vulnerability Note VU#836068 MD5 vulnerable to collision attacks. Vulnerability notes database," CERT Carnegie Mellon University Software Engineering Institute, 2008.

[4] B. E. K. P. A. A. M. Y. Stevens M., "The First Collision for Full SHA-1," in *In: Katz J., Shacham H. (eds) Advances in Cryptology – CRYPTO 2017. CRYPTO 2017. Lecture Notes in Computer Science, vol 10401. Springer*, Cham, 2017.

[5] M. Sprengers, GPU-based Password Cracking, M.Sc. Radboud University Nijmegen Faculty of Science Kerckhoffs Institute, 2011.

[6] HashCat, "https://hashcat.net/hashcat," [Online].

[7] J. t. Ripper, "https://www.openwall.com/john," [Online].

[8] M. S. Esseissah, A. Bhery and H. M. Bahig, "mproving BDD Enumeration for LWE Problem Using GPU," *IEEE Access*, vol. 8, pp. 19737-19749, 2020.

[9] Menezes, A., Van Oorschot, P. C., and Vanstone, S. A., Handbook of Applied Cryptography, CRC Press, 1996.