Performance Comparison of GPU and CPU-based Execution of SHA-3-256 in Regards to Password-Cracking

Salman Ma'arif Achsien (*18221102*)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
smaarif.achsien@gmail.com

Abstract—**Since time immemorial, passwords have been used to ensure the confidentiality of as well as access control towards information. In modern times, the use of passwords is slightly modified, in that they are stored in a scrambled, hash form that is difficult to reverse. This makes password-cracking a difficult and expensive task, especially with the development of secure hashing algorithms such as SHA-3-256. Luckily, advancements in GPU technology have allowed for greater performances. We demonstrate using hashcat that the use of a GPU, as well as the parallel programming paradigm it offers, is superior to that of a CPU when it comes to brute-force password-cracking.**

Keywords—password, password-cracking, SHA-3-256, CPU, GPU, hashcat, performance, benchmark

I. Background

Guaranteeing the confidentiality of information has been a civilization-wide concern ever since the founding of the first empires, and perhaps since even earlier than that. There are many ways of maintaining the confidentiality of information, chief among which is by restricting access to information in the first place. This act of allowing access towards information only to those authorized is known as access control. Access control is then usually complemented with authentication, that is the act of confirming (authenticating) the identity of actors in a system.

There are several methods that may be used to authenticate actors, chief, and most primitive, among these methods are passwords. Traditionally, passwords are words that are ideally known only by those authorized to access confidential information and or restricted systems. When an actor attempts to access the relevant information or system, a challenge will be issued to said actor by asking them for the password. If the actor provides the correct password, their identity will be said to be authenticated, and access will be given.

The use of passwords in maintaining confidentiality and access control have been attested to at the very least since the classical era. Polybius describes how the Roman military employed passwords in order to restrict access to certain areas, most usually the legionary encampment [14]. More recently, in digital systems, passwords have also been used to authenticate logins into user accounts, and consequently maintaining the confidentiality of and access control towards any information or systems that said user has rights to [15].

The way in which passwords are most commonly in digital systems is slightly different to how it is traditionally used. In a traditional scenario, the owner is not the only one with knowledge of the password; instead, those in charge of authenticating said owner would also have knowledge of the password. This method of using passwords has a high chance of causing unauthorized access to information and thus breaking confidentiality and access control, as anyone in possession of the password is able to use it and thus authenticate themselves as the legitimate owner. That is not the case in most digital systems. Instead, those in charge of authentication only have knowledge of the scrambled or hashed form of the password, which is calculated using a one-way function known as a hash function. When an actor answers the authentication challenge and provides the password, this password is first run through the hash function, the results of which are sent to the authenticator. Thus, only the owner knows and is able to use the password, and not even those in charge of authentication have knowledge of it.

Due to the one-way nature of hashing algorithms, passwords stored in a hashed manner are extremely difficult and expensive to recover. This is by design. Given a hash digest calculated using a secure hashing algorithm as well as knowledge of the algorithm used, anyone attempting to recover the scrambled message (hereby referred to as an/the attacker) may only do so by calculating the hash of every possible message and then comparing the results obtained with the desired hash for any (potential) matches. This process of brute-forcing hash results is computationally expensive, as the already expensive hash process must be repeated for every message tried. In this manner are passwords kept confidential to its owner, as any attacker, not even those who by design are in possession of the hash such as websites in the case of user logins, are theoretically able to recover and know its plaintext.

It is to be noted, however, that this difficulty of recovering passwords is a double-edged sword, as there are cases where hashed passwords must be recovered for legitimate, non-malicious purposes. Cases include the recovery of passwords that are forgotten by their owner and are unable to be changed or recovered by other means, as well as the recovery of passwords for investigation or legal purposes. In such cases, the cost of hash-cracking ends up being a burden rather than a gain.

Historically, the time needed to crack passwords is known to be very long. However, several factors have significantly reduced this figure. This includes the general advancement of CPUs, which have led to increased clock speeds and thus speed of hash-calculation. However, and more importantly for this article, this also includes the advancement and proliferation of distributed and parallel computing, brought about by the increasing performance and availability of graphics processing units or GPUs.

By using a parallel programming paradigm as well as the hardware and software necessary to support it (such as the CUDA toolkit for GPUs made by Nvidia), attackers (whether legitimate or malicious) are theoretically able to significantly shorten the time needed to crack passwords. This is because, by doing so, hashes are no longer calculated sequentially one-at-a-time. Rather, several hundred or even thousands of hashes can be calculated simultaneously and parallelly across the myriad cores of the GPU.

Contributions. In this paper, we will attempt to compare the performance of CPUs and GPUs for the purposes of hash-cracking by comparing the benchmark results of a CPU-based implementation of the SHA3-256 hashing algorithm with a GPU-based implementation of said algorithm.

Outline. The rest of the document is outlined as follows. In section II, we describe the research methodology as well as any and all limitations applicable. In section III, we will conduct a literature review that explores any and all relevant theories, concepts, standards, and prior studies. In section IV, we will implement the SHA-3-256 algorithm for use with the CPU and the GPU, conduct a benchmark which evaluates and compares the performance of each in attempting a hash-crack, as well as outline and analyze the results. Lastly, in section V, we will provide general conclusions in regards to the difference in performance (if any), as well as outline any suggestions for further research.

II. Research Methodology

A. Methodology

We will first explore any and all relevant theories, concepts, standards, and prior studies via literature review. Afterwards, we will perform a password-cracking attack on a SHA-3-256 hash using hashcat in both its CPU and GPU modes while benchmarking the performance of each.

In order to evaluate and compare the performance of each, both algorithms will then be given a target hash, as well as a wordlist of passwords that will be tried. The performance will be measured using the time needed to successfully crack the hash, i.e., the time needed to successfully find the string whose hash matches the given hash.

B. Limitations

We limited our research to the exploration of relevant theories, concepts, and prior studies, as well as an experiment in order to determine and compare the performance of SHA-3-256 in each processing unit (CPU and GPU) as well as their respective paradigms. The comparison will be done between hashcat's implementation of SHA-3-256 for both the CPU and GPU. As this research aims to compare the performance of each paradigm for use in password-cracking, no additional optimizations will be made for both implementations unless absolutely necessary.

III. Literature Review

A. Hashing

A hash function is defined as a one-way function that receives a message of any length as input, and produces a value of fixed length as output [3]. The one-way and compressing nature of hash functions have led it to become the backbone of many modern systems and security techniques, ranging from database indexing and cryptocurrencies to message authentication and digital signatures. However, perhaps the most well-known and applied use of hash functions is for the secure and confidential storage of passwords [3].

It is to be noted, however, that not all hash functions are secure. A hash function may only be said to be secure if it fulfills all three of the following properties :

1. For a hash function H and given output of h, it is difficult to find any input x where H(x) = h. This property is known as the (first) preimage resistance property.
2. For a hash function H and given input x, it is difficult to find a second input y where y ≠ x and H(x) = H(y). This property is known as the second preimage resistance property.
3. For a hash function H, It is difficult to find a pair of inputs x ≠ y where H(x) = H(y). This property is known as the collision resistance property. While similar to the second preimage resistance property, and the fact that any violation of the second preimage resistance property also results in the violation of the collision resistance property, this property is different in that no known input is given, and that it only cares about whether or not any collision (the situation in which two different messages produces the same hash value) occurs. [16]

Over the years, many different hashing algorithms have been developed, many of which have been deemed insecure due to the violation of at least one of the above properties. For example, the MD5 algorithm, one of the most well-known and most-used hashing algorithms, has been described as being "cryptographically broken and unsuitable for further use", due to the many weaknesses (one of which being its susceptibility to collisions and thus violation of the collision resistance property) that have been found within it [2]. As such, researchers around the world are constantly developing new algorithms, while also attempting to find weaknesses in already existing ones.

This drive to create "the perfect hashing algorithm" - which not only possesses perfect security but also provides great performance - has, among others, resulted in the creation of various hash construction methods. These construction methods essentially describe how a hash value is created from its input. The following are four well known constructions:

1. Merkle-Damgård or MD construction, which is one of the earliest construction methods and is the one used in the MD-5, SHA-1, and SHA-2 algorithms;
2. Wide Pipe Hash construction, which is created to solve the weaknesses of and improve upon MD construction by having a larger internal state size;
3. Fast Wide Pipe construction, which improves upon the Wide PIpe Hash construction by giving speeds twice as fast; and lastly,
4. Sponge construction, which aims to replace MD construction and works in a fundamentally different manner compared to it. [3]

Out of all construction methods mentioned above, the sponge construction is one of the most novel, and many have proved its robustness. In simple terms, the sponge construction works by first absorbing the messages into the internal state of the hash function, before then squeezing out the bits of the state as output [11]. This construction method is what is used in the SHA-3 family of hashing algorithms, which is the latest in the Secure Hash Algorithm (SHA) family.

B. SHA-3-256

The SHA-3 algorithm is the newest algorithm in the Secure Hash Algorithm (SHA) family of hashing algorithms [6][8][13]. Released by the United States National Institute of Standards and Technology (NIST) in August 2015, the SHA-3 algorithm is based on the Keccak algorithm, which was selected as the winner for the NIST hash function competition . While it is said that SHA-3 is based off of Keccak, it is to be noted that in reality, the underlying algorithm is nearly identical; the only difference being in how a message is padded before its hash is calculated.

There are several instances, or variations, on the SHA-3 algorithm. These variations are generally based on the desired size of the output or hash value, which as defined by NIST, are either 224, 256, 384, or 512 bits long. Thus, the instance where the desired output size is 256 bits long is known as the SHA-3-256 hashing algorithm.

The following is a simplified description of how SHA-3-256 works, as outlined in FIPS (Federal Information Processing Standard) 202 [6][13]:

Preparation phase:

1. Determine and or calculate the following values as the following:
   a. Output size (d), in bits. For SHA-3-256, d = 256 in bits.
   b. Capacity (c) in bits, where c = 2 * d. For SHA-3-256, c = 512 bits.
   c. Rate, also known as block size (r), in bits, where r = 1600 - c. For SHA-3-256, r = 1088 bits.
   d. The length of the message (m) in bits.
2. Calculate the number of padding bytes (q) needed to be appended to the message, where q = (r/8) - (m mod (r/8)). This assumes that the plaintext or input message M is byte-aligned, that is, a multiple of 8 bits long.
3. Append the padding bytes as outlined in Fig. 1. For cases where q > 2, the notation 0x00 symbolizes a string that consists of q - 2 "zero" bytes, i.e, 0x00. repeated q - 2 times.
4. Split the input message M into n blocks of r-size.
5. Initialize an internal state S of 1600 zero bytes, or in other words, a 5x5 array of unsigned 64-bit integers each with a value of zero. The first r-bits of this state is the rate, and the remaining bits are the capacity.

FIGURE I. SHA-3 PADDING

Number of Padding Bytes (q)	Padded Message
1	M || 0x06
2	M || 0x0680
> 2	M || 0x06 || 0x00… || 0x80

Fig. 1. Padding Guide for SHA-3 as per FIPS 202

Absorption phase:

6. Initialize the iteration counter i as 0.
7. XOR the internal state S with the i-th block of the split-up message M_i. This results in the changing of the rate bits, but not the capacity bits.
8. Apply the block permutation function, also known as the Keccak function f, to the internal state S.
9. Increment i, and repeat the absorption phase until i equals the amount of blocks.

Squeezing phase:

10. Initialize the output Z.
11. Append the first d-bits of S to Z. For SHA-3-256, this means that the first 256-bits are to be appended.
12. Return Z.

It is to be noted that the above description is only fit for SHA-3, which has fixed output lengths, and not SHAKE which lets the user freely decide the amount of output bits.

An illustration of the above process is given in Fig. 2.

FIGURE II. SHA-3 FLOWCHART

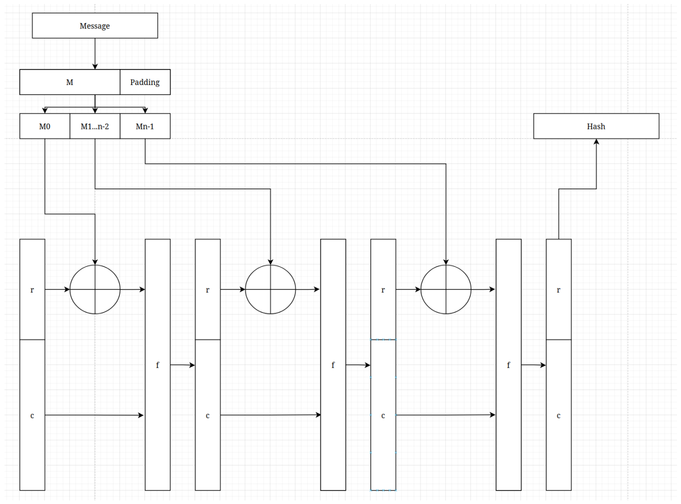


Fig. 2. Flowchart for the SHA-3 Algorithm

In the absorption phase, a certain block permutation function or Keccak function f is mentioned. This consists of 5 predetermined steps or phases, which are named theta, rho, pi, chi, and iota respectively. In the function, these steps are then repeated 24 times, a visual representation of which is given in Fig. 3.

FIGURE III. KECCAK FUNCTION

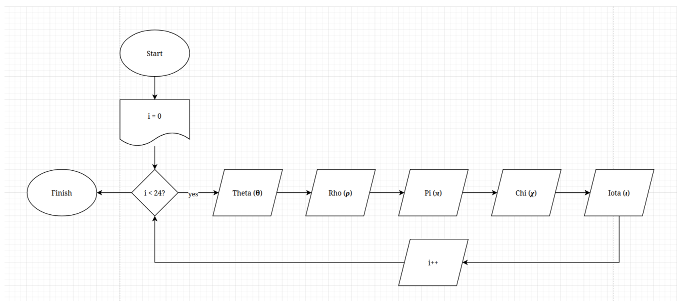


Fig. 3. Flowchart for the Keccak Function

This novel design, as well as its non-use of MD construction unlike the previous algorithms in the SHA family, is what ultimately led to Keccak being selected as the competition winner and becoming SHA-3.

C. *Password Cracking*

As mentioned previously, password cracking can be defined as the process of recovering the original plaintext, that is to say, the unscrambled version, of a password from its hash value [1].

There are many methods or strategies that may be used to crack passwords, with the most common, straightforward, and expensive of which being a brute-force attack. A brute-force attack is defined as an attack in which, given a hash h and hash function H, the attacker guesses all possible passwords until a password x where H(x) = h is found - in other words, until a password that has the same hash value as the given hash is found [1]. This process is computationally very expensive and time consuming, and scales with the difficulty and security of the hashing algorithm being used. Due to the security of SHA-3, the use of this method by attackers is essentially forced.

In a real-world scenario, however, a brute-force attack most likely will not check for every possible permutation. Instead, only permutations from a predefined wordlist will be checked. This word list usually consists of the plaintext of many commonly used passwords, obtained from data breaches of services that either store passwords in plaintext format, or do so using a weak hashing algorithm that is able to or was already reversed quickly. For example, the rockyou.txt wordlist is a word list obtained from the data breach of RockYou, a company which created online social games as well as an application platform for various social networking sites. The company was found to have stored user passwords in plaintext form; this exfiltrated password database was then output into a large text file, and today stands as one of the most commonly used wordlists in brute-force password cracking attacks [12].

In order to assist with hash-cracking, a program known as hashcat may be used. Hashcat is a widely-used password recovery program, with its creators claiming that it is the "world's fastest and most advanced password recovery tool". Hashcat has support for both CPU and GPU [9].

D. *GPUs in Password Cracking*

Compared to CPUs, Graphics Processing Units (also known as GPUs) are designed to perform computations in a parallel, rather than a sequential, manner. That is, calculations are not performed sequentially, but rather parallelly at the same time. This parallel computation allows for more calculations to be performed at once, giving performance increases in cases of repetitive calculations that would normally be blocked by the sequential nature of CPUs [7].

In regards to cryptography, previous studies have shown the drastic performance increases brought about by performing cryptographic calculations in parallel using GPUs. This includes an increase in speeds of RSA and AES encryption when compared to that of CPUs [10], as well as a higher throughput when calculating hashes [4][5].

With this knowledge, it is hypothesized that the usage of GPUs and parallel computation in general, as well as the CUDA platform in particular, may be able to drastically decrease the time needed to perform a brute-force password cracking attack.

IV. ANALYSIS

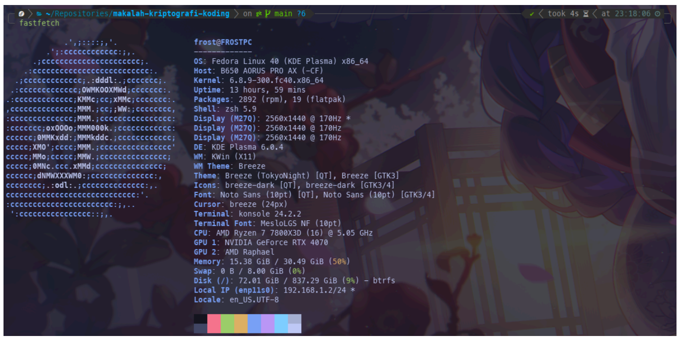
A. *Computer Specifications*

For the purposes of this article, a computer with the following specifications are used:

- OS: Fedora Linux 40 (KDE Plasma)
- CPU: AMD Ryzen 7 7800X3D @ 5.05 GHz
- GPU: NVIDIA GeForce RTX 4070
- RAM: 32 GB DDR5

Proof of the above specifications can be found in Fig. 4

FIGURE IV. COMPUTER SPECIFICATIONS



```
~/Repositories/makalah-kriptografi-koding  on  main ?6                took 4s  at 23:18:06
fastfetch

frost@FROSTPC
-------------
OS: Fedora Linux 40 (KDE Plasma) x86_64
Host: B650 AORUS PRO AX (-CF)
Kernel: 6.8.9-300.fc40.x86_64
Uptime: 13 hours, 59 mins
Packages: 2892 (rpm), 19 (flatpak)
Shell: zsh 5.9
Display (M27Q): 2560x1440 @ 170Hz *
Display (M27Q): 2560x1440 @ 170Hz
Display (M27Q): 2560x1440 @ 170Hz
DE: KDE Plasma 6.0.4
WM: KWin (X11)
WM Theme: Breeze
Theme: Breeze (TokyoNight) [QT], Breeze [GTK3]
Icons: breeze-dark [QT], breeze-dark [GTK3/4]
Font: Noto Sans (10pt) [QT], Noto Sans (10pt) [GTK3/4]
Cursor: breeze (24px)
Terminal: konsole 24.2.2
Terminal Font: MesloLGS NF (10pt)
CPU: AMD Ryzen 7 7800X3D (16) @ 5.05 GHz
GPU 1: NVIDIA GeForce RTX 4070
GPU 2: AMD Raphael
Memory: 15.38 GiB / 30.49 GiB (50%)
Swap: 0 B / 8.00 GiB (0%)
Disk (/): 72.01 GiB / 837.29 GiB (9%) - btrfs
Local IP (enp11s0): 192.168.1.2/24 *
Locale: en_US.UTF-8
```

Fig. 4. Fastfetch Screenshot Displaying Specifications

B. *Hash-cracking Process*

In order to perform a SHA-3-256 password-cracking performance comparison between the CPU and GPU, the hashcat program is used. Each execution will have the –potfile-disable argument, in order to force hashcat to recalculate the hash.

For the GPU, only the OpenCL framework will be used. This is due to an unexpected error regarding the CUDA framework in the writer's computer.

The word list used will be the rockyou.txt wordlist, with the following passwords added (each respective hash also given):

- RaidenEiIsMyWife (5adc36daff1105b8eef714753fcc73df88c8eb6234fa62551f1148a4000a0a4c)
- h0us3ofthehe4rth (b73c1970803f8de5c8693a1a859c6a3aa0a635a35c5e1708b9bde34fe60787ae)
- Cry0byTe! (75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212af74e7fabd5508)

C. *CPU Performance*

In order to benchmark the CPU performance, the following command is used:

```
hashcat -a 0 -m 17400 --opencl-device-types 1 --potfile-disable
--backend-ignore-cuda [hash]  rockyou.txt
```

Running the program three times for each hash, an average speed of 11839.6 kH/s is obtained, with an average time of 0.71ms. The complete results of each execution can be found in Fig. 5, and a sample log and screenshot can be found in Fig. 6 and Fig. 7.

FIGURE V. CPU RESULTS

Password	Speed	Time
RaidenEiIsMyWife	12148.4 kH/s	0.68 ms
h0us3ofthehe4rth	11958.2 kH/s	0.70 ms
Cry0byTe!	11412.3 kH/s	0.74 ms

Fig. 5. CPU Results

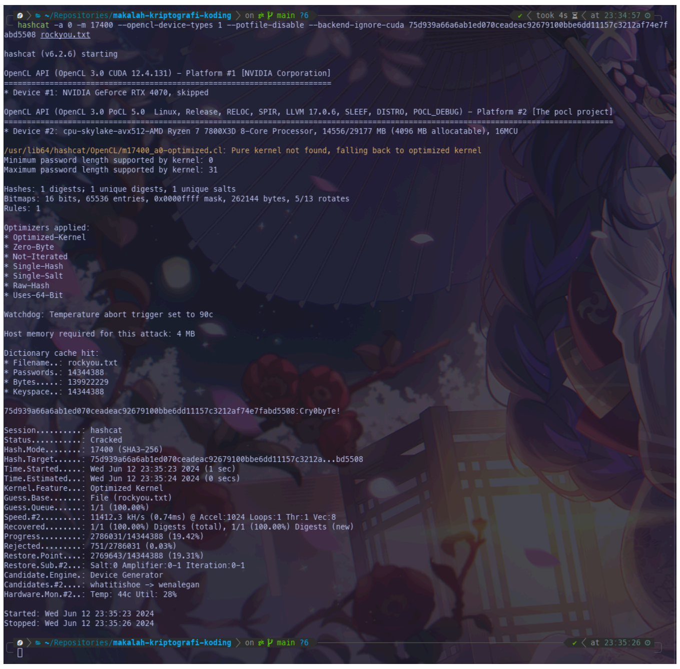
FIGURE VI. CPU SAMPLE OUTPUT

```
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 CUDA 12.4.131) - Platform #1 [NVIDIA
Corporation]
=====================================================
====================
* Device #1: NVIDIA GeForce RTX 4070, skipped

OpenCL API (OpenCL 3.0 PoCL 5.0  Linux, Release, RELOC, SPIR,
LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #2 [The pocl
project]
=====================================================
=====================================================
==============================
* Device #2: cpu-skylake-avx512-AMD Ryzen 7 7800X3D 8-Core
Processor, 14556/29177 MB (4096 MB allocatable), 16MCU

/usr/lib64/hashcat/OpenCL/m17400_a0-optimized.cl: Pure kernel not
found, falling back to optimized kernel
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash
* Uses-64-Bit

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 4 MB

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344388
* Bytes.....: 139922229
* Keyspace..: 14344388

75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212af74e7fabd55
08:Cry0byTe!

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 17400 (SHA3-256)
Hash.Target......:
75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212a...bd5508
Time.Started.....: Wed Jun 12 23:35:23 2024 (1 sec)
Time.Estimated...: Wed Jun 12 23:35:24 2024 (0 secs)
Kernel.Feature...: Optimized Kernel
```

```
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#2.........: 11412.3 kH/s (0.74ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests
(new)
Progress.........: 2786031/14344388 (19.42%)
Rejected.........: 751/2786031 (0.03%)
Restore.Point....: 2769643/14344388 (19.31%)
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#2....: whatitishoe -> wenalegan
Hardware.Mon.#2..: Temp: 44c Util: 28%

Started: Wed Jun 12 23:35:23 2024
Stopped: Wed Jun 12 23:35:26 2024
```

Fig. 6. CPU Sample Output

FIGURE VII. SAMPLE OUTPUT



```
 ~/Repositories/makalah-kriptografi-koding  on  main ?6                    ✔  took 4s  at 23:34:57
  hashcat -a 0 -m 17400 --opencl-device-types 1 --potfile-disable --backend-ignore-cuda 75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212af74e7f
abd5508 rockyou.txt

hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 CUDA 12.4.131) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA GeForce RTX 4070, skipped

OpenCL API (OpenCL 3.0 PoCL 5.0  Linux, Release, RELOC, SPIR, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #2 [The pocl project]
=====================================================================================================================================
* Device #2: cpu-skylake-avx512-AMD Ryzen 7 7800X3D 8-Core Processor, 14556/29177 MB (4096 MB allocatable), 16MCU

/usr/lib64/hashcat/OpenCL/m17400_a0-optimized.cl: Pure kernel not found, falling back to optimized kernel
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash
* Uses-64-Bit

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 4 MB

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344388
* Bytes.....: 139922229
* Keyspace..: 14344388

75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212af74e7fabd5508:Cry0byTe!

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 17400 (SHA3-256)
Hash.Target......: 75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212a...bd5508
Time.Started.....: Wed Jun 12 23:35:23 2024 (1 sec)
Time.Estimated...: Wed Jun 12 23:35:24 2024 (0 secs)
Kernel.Feature...: Optimized Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#2.........: 11412.3 kH/s (0.74ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 2786031/14344388 (19.42%)
Rejected.........: 751/2786031 (0.03%)
Restore.Point....: 2769643/14344388 (19.31%)
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#2....: whatitishoe -> wenalegan
Hardware.Mon.#2..: Temp: 44c Util: 28%

Started: Wed Jun 12 23:35:23 2024
Stopped: Wed Jun 12 23:35:26 2024

 ~/Repositories/makalah-kriptografi-koding  on  main ?6                              ✔  at 23:35:26
```

Fig. 7. Screenshot of Sample CPU Output

D. *GPU Performance*

In order to benchmark the GPU performance, the following command is used:

```
hashcat -a 0 -m 17400 --opencl-device-types 2 --potfile-disable
--backend-ignore-cuda [hash]  rockyou.txt
```

Running the program three times for each hash, an average speed of 12134.1 kH/s is obtained, with an average time of 0.40ms. The complete results of each execution can be found in Fig. 7, and a sample log screenshot can be found in Fig. 8 and Fig. 9.

FIGURE VIII. CPU RESULTS

Password	Speed	Time
RaidenEiIsMyWife	9339.6 kH/s	0.31 ms
h0us3ofthehe4rth	12031.8 kH/s	0.44 ms
Cry0byTe!	15030.9 kH/s	0.44 ms

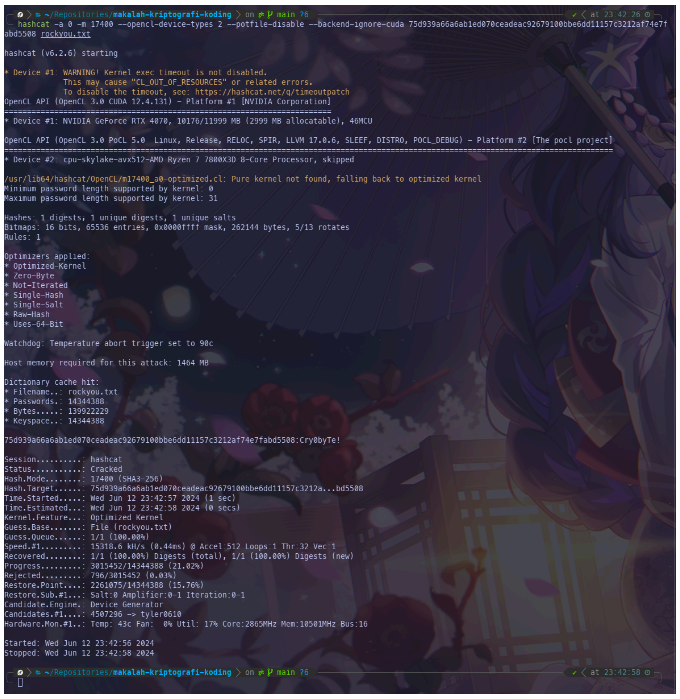
Fig. 8. GPU Results

FIGURE IX. CPU SAMPLE OUTPUT

```
hashcat (v6.2.6) starting

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related
errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
OpenCL API (OpenCL 3.0 CUDA 12.4.131) - Platform #1 [NVIDIA
Corporation]
=================================================================
===================
* Device #1: NVIDIA GeForce RTX 4070, 10176/11999 MB (2999 MB
allocatable), 46MCU

OpenCL API (OpenCL 3.0 PoCL 5.0  Linux, Release, RELOC, SPIR,
LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #2 [The pocl
project]
=================================================================
=================================================================
==============================
* Device #2: cpu-skylake-avx512-AMD Ryzen 7 7800X3D 8-Core
Processor, skipped

/usr/lib64/hashcat/OpenCL/m17400_a0-optimized.cl: Pure kernel not
found, falling back to optimized kernel
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash
* Uses-64-Bit

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1464 MB

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344388
* Bytes.....: 139922229
* Keyspace..: 14344388

75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212af74e7fabd55
08:Cry0byTe!

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 17400 (SHA3-256)
Hash.Target......:
75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212a...bd5508
Time.Started.....: Wed Jun 12 23:42:57 2024 (1 sec)
Time.Estimated...: Wed Jun 12 23:42:58 2024 (0 secs)
Kernel.Feature...: Optimized Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 15318.6 kH/s (0.44ms) @ Accel:512 Loops:1 Thr:32
Vec:1
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests
(new)
Progress.........: 3015452/14344388 (21.02%)
Rejected.........: 796/3015452 (0.03%)
Restore.Point....: 2261075/14344388 (15.76%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 4507296 -> tyler0610
Hardware.Mon.#1..: Temp: 43c Fan:  0% Util: 17% Core:2865MHz
Mem:10501MHz Bus:16

Started: Wed Jun 12 23:42:56 2024
Stopped: Wed Jun 12 23:42:58 2024
```

Fig. 9. CPU Sample Output

FIGURE X. SAMPLE OUTPUT



```
 ~/Repositories/makalah-kriptografi-koding  on  main ?6                                                   ✓  at 23:42:26
 hashcat -a 0 -m 17400 --opencl-device-types 2 --potfile-disable --backend-ignore-cuda 75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212af74e7f
abd5508 rockyou.txt

hashcat (v6.2.6) starting

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
OpenCL API (OpenCL 3.0 CUDA 12.4.131) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA GeForce RTX 4070, 10176/11999 MB (2999 MB allocatable), 46MCU

OpenCL API (OpenCL 3.0 PoCL 5.0  Linux, Release, RELOC, SPIR, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #2 [The pocl project]
=======================================================================================================================================
* Device #2: cpu-skylake-avx512-AMD Ryzen 7 7800X3D 8-Core Processor, skipped

/usr/lib64/hashcat/OpenCL/m17400_a0-optimized.cl: Pure kernel not found, falling back to optimized kernel
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash
* Uses-64-Bit

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1464 MB

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344388
* Bytes.....: 139922229
* Keyspace..: 14344388

75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212af74e7fabd5508:Cry0byTe!

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 17400 (SHA3-256)
Hash.Target......: 75d939a66a6ab1ed070ceadeac92679100bbe6dd11157c3212a...bd5508
Time.Started.....: Wed Jun 12 23:42:57 2024 (1 sec)
Time.Estimated...: Wed Jun 12 23:42:58 2024 (0 secs)
Kernel.Feature...: Optimized Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 15318.6 kH/s (0.44ms) @ Accel:512 Loops:1 Thr:32 Vec:1
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 3015452/14344388 (21.02%)
Rejected.........: 796/3015452 (0.03%)
Restore.Point....: 2261075/14344388 (15.76%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 4507296 -> tyler0610
Hardware.Mon.#1..: Temp: 43c Fan:  0% Util: 17% Core:2865MHz Mem:10501MHz Bus:16

Started: Wed Jun 12 23:42:56 2024
Stopped: Wed Jun 12 23:42:58 2024

 ~/Repositories/makalah-kriptografi-koding  on  main ?6                                                   ✓  at 23:42:58
```

Fig. 10. Screenshot of Sample GPU Output

V. CONCLUSION

Using hashcat, we have successfully proven that cracking a SHA-3-256 hashed password using the GPU is significantly faster than the CPU.

In regards to future research, much can still be done. For one, other APIs, such as CUDA, may also be used. Additionally, a larger as well as more random wordlist could also be used to truly highlight any differences in performance.

ACKNOWLEDGMENT

We, the writer of this article, would like to provide great acknowledgement to the following parties:

1. God Almighty, who created this universe and all of its mathematical and physical laws with the greatest perfection;
2. Dr. Rinaldi Munir M.T., who without his lessons on cryptography would not have allowed us to write this article;
3. Zhillan Attarizal Rezyarifin, who, despite our many differences, has been a great inspiration both outside and inside the context of the this article; and
4. Raiden Ei/Shogun, Yae Miko, Clorinde, Lisa, Peruere (A.K.A. Arlecchino), Kafka, Ceres Fauna, Nerissa Ravencroft, Kurohime, and Makima, who have provided us with great company throughout the writing of this article.

REFERENCES

[1] F. Yu and Y. Huang, "An Overview of Study of Passowrd Cracking," in 2015 International Conference on Computer Science and Mechanical Automation (CSMA), Oct. 2015, pp. 25–29. doi: 10.1109/CSMA.2015.12.

[2] "CERT/CC Vulnerability Note VU#836068." Accessed: Jun. 12, 2024. [Online]. Available: https://www.kb.cert.org

[3] A. A. Alkandari, I. F. Al-Shaikhli, and M. A. Alahmad, "Cryptographic Hash Function: A High Level View," in 2013 International Conference on Informatics and Creative Multimedia, Sep. 2013, pp. 128–134. doi: 10.1109/ICICM.2013.29.

[4] W.-K. Lee, K. Jang, G. Song, H. Kim, S. O. Hwang, and H. Seo, "Efficient Implementation of Lightweight Hash Functions on GPU and Quantum Computers for IoT Applications," IEEE Access, vol. 10, pp. 59661–59674, 2022, doi: 10.1109/ACCESS.2022.3179970.

[5] H. Choi and S. C. Seo, "Fast Implementation of SHA-3 in GPU Environment," IEEE Access, vol. 9, pp. 144574–144586, 2021, doi: 10.1109/ACCESS.2021.3122466.

[6] "FIPS 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions | CSRC." Accessed: Jun. 12, 2024. [Online]. Available: https://csrc.nist.gov/pubs/fips/202/final

[7] R. S. Dehal, C. Munjal, A. A. Ansari, and A. S. Kushwaha, "GPU Computing Revolution: CUDA," in 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Oct. 2018, pp. 197–201. doi: 10.1109/ICACCCN.2018.8748495.

[8] I. T. L. Computer Security Division, "Hash Functions | CSRC | CSRC," CSRC | NIST. Accessed: Jun. 12, 2024. [Online]. Available: https://csrc.nist.gov/projects/hash-functions

[9] "hashcat [hashcat wiki]." Accessed: Jun. 12, 2024. [Online]. Available: https://hashcat.net/wiki/doku.php?id=hashcat

[10] K. Grace, M. S. G. Devasena, and M. Shanmugam, "Joy of GPU Computing: A Performance Comparison of AES and RSA in GPU and CPU," 2020, pp. 425–434. doi: 10.1007/978-981-15-5558-9_39.

[11] "Keccak Team." Accessed: Jun. 12, 2024. [Online]. Available: https://keccak.team/sponge_duplex.html

[12] N. Cubrilovic, "RockYou Hack: From Bad To Worse," TechCrunch. Accessed: Jun. 12, 2024. [Online]. Available: https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/

[13] M. J. Dworkin, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” National Institute of Standards and Technology, NIST FIPS 202, Jul. 2015. doi: 10.6028/NIST.FIPS.202.

[14] Polybius, The Histories, vol. 3. in Loeb Classical Library, no. 138, vol. 3. Harvard University Press, 2011.

[15] R. A. Grimes, “Types of Authentication,” in Hacking Multifactor Authentication, Wiley, 2021, pp. 59–99. doi: 10.1002/9781119672357.ch3.

[16] M. Naor and M. Yung, “Universal one-way hash functions and their cryptographic applications,” in Proceedings of the twenty-first annual ACM symposium on Theory of computing, in STOC ’89. New York, NY, USA: Association for Computing Machinery, Feb. 1989, pp. 33–43. doi: 10.1145/73007.73011.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

[illegible]

Salman Ma'arif Achsien
NIM: 18221102