Analisis Keamanan dan Performa Algoritma Hash SHA-1 dan SHA-3 dalam Skema Autentikasi

Siti Adira Ramadhani (18220094)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
diraramadhani03@gmail.com

Abstract—Password **merupakan komponen yang sering digunakan pada sistem informasi di era *modern* ini. *Password* pun menjadi hal yang penting bagi setiap pengguna karena digunakan sebagai syarat autentikasi. Sistem informasi melindungi *password* dengan melakukan *hash* terlebih dahulu sebelum disimpan ke *database*. Tujuannya adalah menjaga kerahasiaan *password* agar *password* tidak dapat dibaca secara langsung. Dalam tulisan ini, dilakukan analisis dan perbandingan antara dua algoritma hashing SHA-1 dan SHA-3 pada aspek keamanan dan performa pada proses autentikasi.**

Keywords—password; autentikasi; hash; SHA-1; SHA-3; perbandingan

I. Pendahuluan

Perkembangan teknologi juga turut berdampak pada peningkatan jumlah sistem informasi. Keamanan informasi merupakan salah satu komponen penting pada sistem informasi. Mekanisme autentikasi yang biasanya digunakan dalam sistem informasi adalah *username* dan *password*. *Username* bersifat publik, sedangkan *password* bersifat rahasia. *Password* digunakan sebagai komponen krusial yang memberikan akses terhadap sistem informasi.

Salah satu cara untuk menjaga kerahasiaan *password* yang disimpan adalah algoritma *hashing.* Algoritma *hashing* akan mengubah *password* ke dalam nilai unik dengan panjang *output* yang sama. Algoritma *hash* bersifat satu arah, sehingga nilainya tidak dapat dikembalikan ke dalam nilai aslinya. Hal ini menyebabkan *password* tidak dapat dibaca secara langsung.

Terdapat banyak algoritma *hash*, antara lain SHA-1 dan SHA-3. Oleh karena itu, perlu adanya analisis dan perbandingan antara keduanya dalam hal keamanan dan performa dalam konteks autentikasi.

II. Metode

A. Studi literatur

Sebelum memulai penelitian, penulis melakukan studi literatur dari jurnal, materi perkuliahan, dan video *youtube* mengenai autentikasi, fungsi *hash*, algoritma SHA-1, dan algoritma SHA-3.

B. Eksperimen

Setelah itu, penulis melakukan eksperimen dengan membuat *file* Python untuk menganalisis performa *algoritma* dan menggunakan *tool* Hashcat untuk menganalisis keamanan. Dalam menganalisis performa algoritma, akan digunakan beberapa *library* pada *file* Python, yaitu : *hashlib, psutil,* dan *time.* Karena pada *library* terdapat banyak varian untuk algoritma SHA-3, pada kesempatan ini, penulis menggunakan varian sha3_256 yang merupakan varian SHA-3 dengan panjang *message digest* 256 bit.

Pengujian keamanan algoritma *hash* akan menggunakan *tool* yang bernama Hashcat. Metode penyerangan akan dilakukan pada *message digest* dengan mode penyerangan *brute force.*

III. Dasar Teori

A. Autentikasi

Autentikasi adalah proses untuk memverifikasi dan memvalidasi identitas pengguna atau entitas untuk memberikan hak akses ke suatu sistem informasi. Dalam proses autentikasi, sistem akan melakukan pemeriksaan terhadap informasi yang diberikan oleh pengguna dan informasi yang tersimpan. Jika informasi tersebut cocok, maka pengguna atau entitas akan diberikan akses ke sistem, begitu pula sebaliknya. Terdapat tiga faktor autentikasi yang umum digunakan :

1. Faktor pengetahuan
   Merupakan sesuatu yang kita ketahui, misalnya *password* dan PIN.
2. Faktor kepemilikan
   Merupakan sesuatu yang kita miliki, misalnya kartu identitas dan token autentikasi.
3. Faktor melekat
   Merupakan sesuatu yang berada pada diri Anda, misalnya sidik jari, wajah, dan iris mata.

Autentikasi dengan menggunakan *password* merupakan salah satu metode yang paling sering ditemukan pada sistem informasi. Dalam autentikasi dengan *password*, pengguna atau entitas diminta untuk memasukkan kombinasi karakter yang hanya diketahui oleh mereka sebagai bukti identitas. Berikut merupakan gambar proses

autentikasi dengan menggunakan *password* pada sistem informasi.

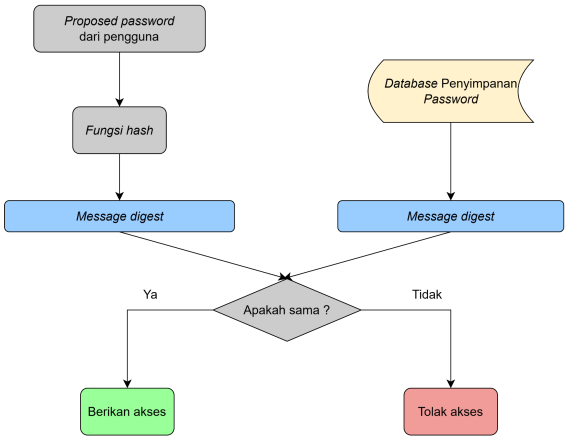

Fig. 1. Ilustrasi Proses Autentikasi dengan *Password* (sumber : A Review Of One Time Password Mobile Verification)

Pertama-tama, pengguna atau entitas akan memasukkan informasi berupa *username* dan *password* sebagai syarat yang diberikan sistem. Kemudian, *password* yang diberikan pengguna akan di-*hash* oleh sistem menggunakan algoritma yang digunakan sistem. Lalu, sistem akan membandingkan nilai hasil *hash* dari *password* yang diberikan pengguna dan nilai *hash* yang tersimpan pada *database* untuk *username* yang dimasukkan. Jika nilainya sama, pengguna akan diberikan akses ke sistem. Jika nilainya berbeda, pengguna tidak diberikan akses ke sistem.

B. *Fungsi Hash*

Fungsi hash adalah fungsi yang mengkompresi suatu pesan berukuran sembarang menjadi suatu pesan ringkas berukuran tetap. Fungsi *hash* yang dibahas pada makalah ini adalah fungsi yang bersifat satu arah, sehingga nilai keluaran *hash* tidak dapat dibalikkan ke pesan semula. Persamaan umum fungsi *hash* dapat direpresentasikan sebagai fungsi berikut.

$$h = H(M)$$

Secara matematis, fungsi *hash* dapat dinyatakan sebagai berikut.

$$h_i = H(M_i, h_{i-1})$$

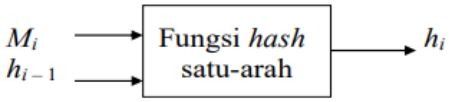

Fig. 2. Skema Fungsi *Hash* (sumber : Slide Kuliah Fungsi Hash)

Pesan yang panjangnya melebihi ukuran blok *hash* akan dibagi menjadi blok-blok yang lebih kecil. Setiap blok pesan kemudian diolah secara berurutan menggunakan fungsi *hash* untuk menghasilkan nilai *hash* dengan masukan berupa blok pesan (M_i) dan nilai *hashing* blok pesan sebelumnya (h_{i-1}). Setelah semua blok pesan diproses, algoritma *hash* menggabungkan hasil-hasil *hash* dari setiap blok pesan menjadi sebuah nilai *hash* akhir yang disebut *message digest.*

Terdapat banyak algoritma fungsi *hash* satu arah, antara lain algoritma *hash* varian MD, algoritma *hash* varian SHA, dan algoritma *hash* WHIRPOOL. Di antara varian algoritma-algoritma tersebut, terdapat beberapa algoritma yang sudah jarang digunakan karena telah dianggap kurang aman.

1) *Algoritma SHA-1*

SHA-1 adalah merupakan salah satu bagian dari varian fungsi *hash* satu arah *Secure Hash Algorithm (SHA)*. SHA-1 ditemukan oleh NIST. SHA didasarkan pada *MD4* yang dibuat oleh Ronald L. Rivest dari MIT. Algoritma SHA-1 menerima masukan dengan ukuran maksimum 2^{64} bit dan menghasilkan *message digest* sepanjang 160 bit.

Berikut ini merupakan langkah-langkah dari algoritma SHA-1 untuk menghasilkan *message digest*.

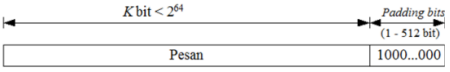

Fig. 3. Skema Proses Inisiasi *Padding Bit* Algoritma SHA-1 (sumber : Slide Kuliah Fungsi Hash SHA-1)

1. Pada awalnya, akan ditambahkan bit-bit pengganjal *(padding bits)* dengan panjang 1 - 512 bit sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 (mod 512). Bit-bit pengganjal terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

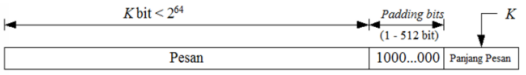

Fig. 4. Skema Proses Penambahan *Bit* Informasi Panjang Pesan Algoritma SHA-1 (sumber : Slide Kuliah Fungsi Hash SHA-1)

2. Kemudian, akan ditambahkan nilai panjang pesan semula yang dinyatakan dalam 64 bit. Oleh karena itu, panjang pesan pada tahap ini merupakan kelipatan 512.
3. Setelah itu, dilakukan inisialisasi penyangga (*buffer*) MD sejumlah 5 penyangga dengan panjang setiap penyangganya adalah 32 bit. Nilai penyangga ini bersifat tetap dan diberi nama A, B, C, D, dan E dengan nilai sebagai berikut.
    a. A = 67452301
    b. B = EFCDAB89
    c. C = 98BADCFE
    d. D = 10325476
    e. E = C3D2E1F0

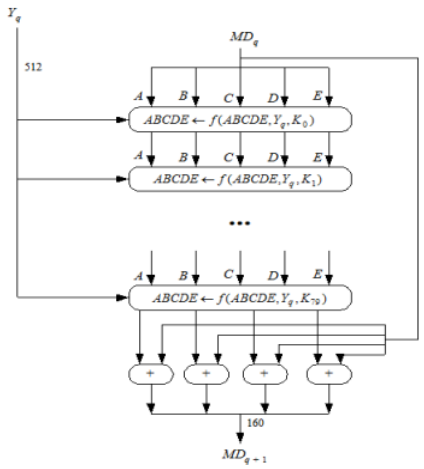

Fig. 5. Skema Proses Menghasilkan *Message Digest* pada Algoritma SHA-1 (sumber : Slide Kuliah Fungsi Hash SHA-1)

4. Lalu, pesan tersebut akan diolah dalam blok berukuran 512 bit dan melalui proses sebanyak 80 putaran. Setiap putaran menggunakan bilangan penambah K_t , dengan rincian sebagai berikut.
    a. Putaran $0 < t < 19$ K_t = 5A827999
    b. Putaran $20 < t < 39$ K_t = 6ED9EBA1
    c. Putaran $40 < t < 59$ K_t = 8F1BBCDC
    d. Putaran $60 < t < 79$ K_t = CA62C1D6

2) Algoritma SHA-3

Setelah beberapa tahun, NIST menyelenggarakan kompetisi terbuka untuk mengembangkan fungsi *hash* baru sebagai komplementer SHA-1 dan SHA-2. Kompetisi ini diumumkan tahun 2007 dan berakhir pada Oktober 2012. Pada babak final, terdapat 5 finalis, yaitu : BLAKE, Grøstl, JH, Keccak, dan Skein. Di antara finalis tersebut, pemenang kompetisi ini adalah Keccak, sehingga Keccak akan menjadi cikal bakal SHA-3.

Nama "Keccak" berasal dari kata "Kecak", sebuah tari dari Bali. Desain algoritma Keccak berbeda dari finalis SHA-3 lainnya yang bergantung pada fungsi kompresi. Dalam desainnya, Keccak menggunakan konstruksi "spons" dan "memeras".

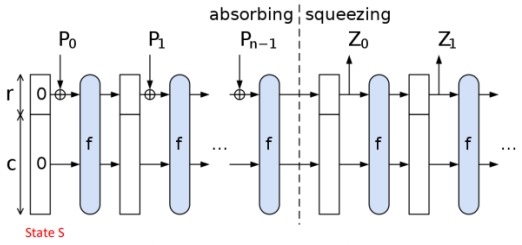

Fig. 6. Skema Proses Menghasilkan *Message Digest* pada Algoritma SHA-3 (sumber : Slide Kuliah Fungsi Hash SHA-3)

Berikut ini merupakan langkah-langkah dari algoritma SHA-3 untuk menghasilkan *message digest*.

1. Fase pra proses
    a. Misalkan panjang *digest* yang diinginkan adalah d-bit
    b. Pertama, pesan M ditambah dengan bit-bit pengganjal *(padding)* menjadi *string* P sehingga panjang P habis dibagi dengan r atau n = panjang(P)/r
    c. Selanjutnya, P dipotong menjadi blok-blok P_i berukuran r-bit
    d. Kemudian, b-bit dari peubah status (*state*) S diinisialisasi menjadi nol (b = r + c).
2. Fase penyerapan *(absorbing)*
    a. Untuk setiap blok masukan Pi berukuran r-bit, XOR-kan dengan r-bit pertama dari state S, lalu hasilnya dimasukkan ke dalam fungsi permutasi f untuk menghasilkan *state* baru S.
    b. Bila semua blok masukan selesai diproses, konstruksi spons beralih ke fase pemerasan *(squeezing)*
3. Fase pemerasan *(squeezing)*
    a. *Message digest* akan disimpan di dalam Z.
    b. Inisialisasi Z dengan *string* kosong (null string).
    c. Selagi panjang Z belum sama dengan d, r-bit pertama dari state S disambungkan *(append)* ke Z.
    d. Jika panjang Z masih belum sama dengan d, masukkan ke dalam
    e. fungsi permutasi f menghasilkan *state* baru S.
    f. Sebanyak c bit terakhir dari *state* tidak pernah terpengaruh secara langsung oleh blok masukan dan tidak pernah mengeluarkan luaran selama fase pemerasan. Hal ini untuk mencegah terjadinya kolisi

C. Serangan *Brute Force*

Serangan *brute force* merupakan metode serangan dengan mencoba semua kemungkinan kombinasi secara sistematis untuk mendapatkan akses tanpa izin ke suatu sistem atau entitas yang terproteksi. Dalam konteks keamanan komputer, serangan *brute force* sering digunakan untuk mencoba menebak kata sandi atau kunci enkripsi dengan mencoba semua kemungkinan kombinasi.

Apabila berhasil menebak kata sandi atau kunci enkripsi dengan benar, penyerang dapat mendapatkan akses tidak sah ke akun korban, data pribadi korban, informasi keuangan korban, atau data lainnya. Dalam beberapa kasus, serangan *brute force* juga dapat digunakan untuk membobol sistem keamanan dan mendapatkan akses ke data penting, termasuk data perusahaan atau pemerintah yang rahasia.

IV. RANCANGAN

Seluruh pengujian dilakukan pada perangkat pribadi penulis. Berikut merupakan spesifikasi lingkungan pengujian yang dilakukan dalam menyusun makalah ini.

TABLE I. SPESIFIKASI LINGKUNGAN PENGUJIAN

Nama spesifikasi	Nilai
Processor	Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
RAM	8,00 GB
Operating system	Windows 11 Home Single Language
versi Python	Python 3.10.6
versi *Library*	hashlib : *library* standar Python time : *library* standar Python psutil : 5.9.5
versi Hashcat	6.2.6

(sumber: Hasil Pengujian)

Analisis perbandingan performa antara kedua algoritma meliputi perbandingan nilai rata-rata *execution time*, *CPU usage*, dan *memory usage* dalam 1000 putaran. Analisis ini akan menggunakan program Python dengan alur sebagai berikut.

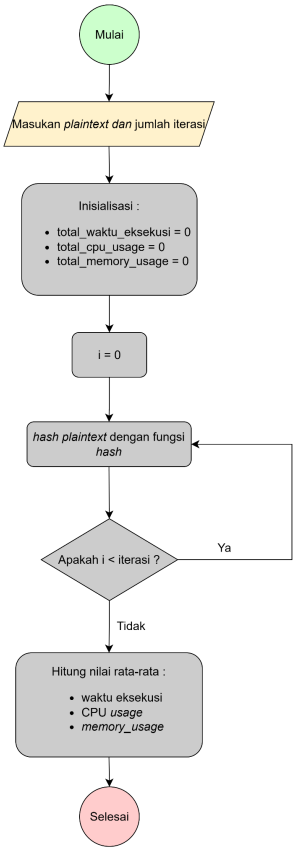

Fig. 7. *Flowchart* Pengujian Performa Algoritma *Hash* (sumber : Dokumentasi pribadi)

Analisis perbandingan keamanan antara kedua algoritma akan menggunakan *tool* berupa Hashcat. Pada skenario pengujian keamanan ini, akan dilakukan serangan *brute force* terhadap *message digest* dari kata sandi dengan panjang delapan hingga dua belas karakter. Tujuannya adalah untuk mengukur waktu *real* atau estimasi waktu yang dibutuhkan untuk menemukan *plaintext* dari sebuah *message digest* saat serangan *brute force* dilakukan. Adapun perintah yang digunakan untuk menjalankan serangan *brute force* pada *tool* Hashcat adalah sebagai berikut.

```
hashcat -m <kode_algoritma> -a 3 -o
<nama_file_output>.txt <nama_file_hasht>.txt
<guess_mask>
```

Keterangan :

1. hashcat
   perintah untuk menjalankan program Hashcat.
2. -m <kode_algoritma>
   parameter untuk memberitahukan nama algoritma *hash* yang digunakan. Kode untuk SHA-1 adalah 100 dan kode untuk SHA-3 256 adalah 17400.
3. -a 3
   parameter untuk menggunakan metode *brute force.*
4. -o <nama_file_output>.txt
   parameter untuk menuliskan hasil *cracked* ke nama *file* teks yang di-*assign.*
5. <nama_file_hash>.txt
   file teks yang menyimpan nilai *hash* dari algoritma SHA-1 dan SHA-3.
6. <guess_mask>
   merupakan *mask* untuk memberitahukan dugaan jenis karakter untuk tiap karakter pada *plaintext. Mask* yang dapat digunakan adalah sebagai berikut.
   a. ?l : *lowercase*
   b. ?u: *uppercase*
   c. ?d :decimal
   d. ?s : simbol

V. IMPLEMENTASI DAN PENGUJIAN

A. *Implementasi*

Program pengujian performa algoritma *hash* SHA-1 dan SHA-3 (*message digest* : 256 bit) diimplementasikan menggunakan bahasa pemrograman Python. Berikut merupakan kode program pengujian performa untuk setiap algoritma *hash.*

Program Pengujian Performa SHA-1

```
import hashlib
import psutil
import time

def calculate_performance_sha1(message,
iterations):
    total_execution_time = 0
    total_cpu_usage = 0
    total_memory_usage = 0

    for i in range(iterations):
        # start
        start_time = time.time()
        start_cpu_usage = psutil.cpu_percent()
        start_memory_usage =
psutil.Process().memory_info().rss
```

```
        # Hashing
        hash_object =
hashlib.sha1(message.encode())
        hex_digest = hash_object.hexdigest()

        # end
        end_time = time.time()
        end_cpu_usage = psutil.cpu_percent()
        end_memory_usage =
psutil.Process().memory_info().rss

        # calculate
        execution_time = end_time - start_time
        cpu_usage = max(end_cpu_usage -
start_cpu_usage, 0) # set negative value to 0
        memory_usage = end_memory_usage -
start_memory_usage

        total_execution_time += execution_time
        total_cpu_usage += cpu_usage
        total_memory_usage += memory_usage


    average_execution_time =
(total_execution_time / iterations) * 1000 #
convert to milliseconds
    average_cpu_usage = (total_cpu_usage /
iterations)
    average_memory_usage = (total_memory_usage
/ iterations)

    print('SHA-1 hash result:', hex_digest)
    print('Average execution time:',
average_execution_time, 'milliseconds')
    print('Average CPU usage:',
average_cpu_usage, '%')
    print('Average memory usage:',
average_memory_usage, 'bytes')

message = input("Masukkan password : ")
print(f'Panjang password : {len(message)}')
calculate_performance_sha1(message, 1000)
```

Program Pengujian Performa SHA-3 dengan *message digest* 256 bit

```
import hashlib
import psutil
import time

def calculate_performance_sha3_256(message,
iterations):
    total_execution_time = 0
    total_cpu_usage = 0
    total_memory_usage = 0

    for i in range(iterations):
        # start
        start_time = time.time()
        start_cpu_usage = psutil.cpu_percent()
        start_memory_usage =
psutil.Process().memory_info().rss

        # Hashing
        hash_object =
hashlib.sha3_256(message.encode())
        hex_digest = hash_object.hexdigest()

        # end
        end_time = time.time()
        end_cpu_usage = psutil.cpu_percent()
        end_memory_usage =
psutil.Process().memory_info().rss

        # calculate
        execution_time = end_time - start_time
        cpu_usage = max(end_cpu_usage -
start_cpu_usage, 0)
        memory_usage = end_memory_usage -
start_memory_usage

        total_execution_time += execution_time
        total_cpu_usage += cpu_usage
        total_memory_usage += memory_usage

    average_execution_time =
(total_execution_time / iterations) * 1000 #
convert to milliseconds
    average_cpu_usage = (total_cpu_usage /
iterations)
    average_memory_usage = (total_memory_usage
/ iterations)

    print('SHA-3 hash result:', hex_digest)
    print('Average execution time:',
average_execution_time, 'milliseconds')
    print('Average CPU usage:',
average_cpu_usage, '%')
    print('Average memory usage:',
average_memory_usage, 'bytes')


message = input("Masukkan password : ")
print(f'Panjang password : {len(message)}')
calculate_performance_sha3_256(message, 1000)
```

B. Pengujian

1) Performa

Hasil pengujian performa terdiri dari tiga bagian, yaitu waktu rata-rata *execution time*, rata-rata *CPU usage*, dan rata-rata *memory usage* dengan iterasi 1000 putaran. Berikut merupakan hasil pengujian rata-rata *execution time* dari algoritma SHA-1 dan SHA-3 (*message digest* : 256 bit) dalam satuan milidetik.

TABLE II. PERBANDINGAN RATA-RATA EXECUTION TIME SHA-1 DAN SHA-3 (MESSAGE DIGEST : 256 BIT)

Panjang karakter	Rata-Rata *Execution time (milidetik)*			Algoritma yang lebih unggul
	SHA-1	*SHA-3 (message digest : 256 bit)*	Selisih	
8	0,05200	0,06964	0,01764	SHA-1
9	0,05274	0,06436	0,01162	SHA-1
10	0,05377	0,06920	0,01543	SHA-1
11	0,06158	0,06405	0,00247	SHA-1
12	0,06415	0,07091	0,00676	SHA-1

(sumber: Hasil Pengujian)

Berikut merupakan hasil pengujian rata-rata *CPU usage* dari algoritma SHA-1 dan SHA-3 (*message digest* : 256 bit) dalam persen (%).

TABLE III. PERBANDINGAN RATA-RATA CPU USAGE SHA-1 DAN SHA-3 (MESSAGE DIGEST : 256 BIT)

Panjang karakter	Rata-Rata *CPU usage (%)*			Algoritma yang lebih unggul
	SHA-1	*SHA-3 (message digest : 256 bit)*	Selisih	
8	0,115	0,139	0,024	SHA-1
9	0,074	0,069	0,005	SHA-3
10	0,142	0,148	0,006	SHA-1
11	0,127	0,205	0,078	SHA-1
12	0,161	0,300	0,139	SHA-1

(sumber: Hasil Pengujian)

Berikut merupakan hasil pengujian rata-rata *memory usage* dari algoritma SHA-1 dan SHA-3 (*message digest* : 256 bit) dalam satuan *bytes*.

TABLE IV. PERBANDINGAN RATA-RATA MEMORY USAGE SHA-1 DAN SHA-3 (MESSAGE DIGEST : 256 BIT)

Panjang karakter	Rata-Rata *Memory usage (bytes)*			Algoritma yang lebih unggul
	SHA-1	*SHA-3 (message digest : 256 bit)*	Selisih	
8	38,502	31,130	7,372	SHA-3
9	37,683	31,949	5,734	SHA-3
10	38,502	31,130	7,372	SHA-3
11	40,141	32,768	7,373	SHA-3
12	39,322	30,310	9,012	SHA-3

(sumber: Hasil Pengujian)

2) Keamanan

Terdapat dua jenis hasil dari pengujian keamanan ini, yaitu waktu *real* dan estimasi waktu penyerangan *brute force*. Waktu *real* adalah waktu yang diperlukan dalam proses penyerangan *brute force* hingga diperoleh *plaintext* asli. Waktu *real* akan digunakan untuk menguji *plaintext* sepanjang 8 karakter. Namun, *plaintext* dengan panjang 9 hingga 12 karakter hanya akan menggunakan estimasi waktu untuk melakukan serangan *brute force*. Hal ini disebabkan oleh proses serangan *brute force* dari *plaintext* yang lebih panjang dari delapan karakter membutuhkan waktu yang lama.

```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 100 (SHA1)
Hash.Target......: a22aa1e2b3f5381f0ee2532615400c749924d440
Time.Started.....: Mon May 22 10:47:47 2023 (31 secs)
Time.Estimated...: Mon May 22 10:48:18 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?l?l?d?l?l?l?l?l [8]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   721.6 MH/s (8.38ms) @ Accel:64 Loops:1024 Thr:32 Vec:1
Speed.#2.........:   101.0 MH/s (7.53ms) @ Accel:8 Loops:16 Thr:256 Vec:1
Speed.#*.........:   822.5 MH/s
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 25241763840/80318101760 (31.43%)
Rejected.........: 0/25241763840 (0.00%)
Restore.Point....: 3231744/11881376 (27.20%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1024 Iteration:0-1024
Restore.Sub.#2...: Salt:0 Amplifier:2192-2208 Iteration:0-16
Candidate.Engine.: Device Generator
Candidates.#1....: sa1vsuct -> tj2jdfuc
Candidates.#2....: dy4vatqt -> qp3iqxqt
Hardware.Mon.#1..: Temp: 77c Util: 94% Core:1594MHz Mem:3003MHz Bus:4
Hardware.Mon.#2..: N/A
```

Fig. 8. Hasil Pengujian Serangan *Brute Force* dengan Hashcat (sumber : Dokumentasi Pribadi)

Berikut merupakan *test case* dan hasil waktu *real* atau waktu estimasi untuk melakukan serangan *brute force* dengan *tool* Hashcat.

TABLE V. PERBANDINGAN WAKTU DAN ESTIMASI WAKTU PADA BRUTE FORCE TESTING SHA-1 DAN SHA-3 (MESSAGE DIGEST : 256 BIT)

Target Plaintext	Panjang karakter	Waktu Serangan		Algoritma yang lebih unggul
		SHA-1	*SHA-3 (message digest : 256 bit)*	
my2girls	8	31 detik	3 menit 47 detik	SHA-3
Sunshine1	9	17 menit	1 jam 53 menit	SHA-3
Godisgood1	10	19 menit	5 hari 9 jam	SHA-3
iloveyou300	11	3 hari 2 jam	20 hari 16 jam	SHA-3
tinkerbell12	12	209 hari 3 jam	3 tahun 325 hari	SHA-3

(sumber: Hasil Pengujian)

VI. KESIMPULAN

Dalam aspek *execution time* dan *CPU usage*, algoritma SHA-1 lebih unggul. Namun, selisih antara keduanya sangat kecil. Oleh karena itu, dapat dikatakan perbedaan waktu dan penggunaan *CPU* tidak akan terasa secara nyata.

Dalam aspek *memory usage* dan waktu yang dibutuhkan dalam serangan *brute force,* algoritma SHA-3 lebih unggul. Selisih *memory usage* antara keduanya memang sangat kecil, bahkan tidak mencapai 10 *bytes*. Akan tetapi, selisih waktu untuk melakukan serangan *brute force attack* antara keduanya sangat besar dengan waktu SHA-3 yang jauh lebih lama. Hal ini menunjukkan SHA-3 memiliki tingkat keamanan yang jauh lebih tinggi.

Berdasarkan hal tersebut, dapat disimpulkan bahwa algoritma SHA-3 lebih baik dibandingkan dengan SHA-1.

Hal ini dapat dilihat melalui keunggulannya pada aspek *memory usage* dan keamanan terhadap serangan *brute force* yang lebih unggul, walaupun performa dari *execution time* dan *CPU usage* sedikit lebih rendah.Oleh karena itu, akan lebih baik. Oleh karena itu, penggunaan algoritma SHA-3 untuk skema autentikasi dan keamanan *password* lebih disarankan dibandingkan dengan SHA-1.

Link Source Code

https://github.com/meandmyabsurdlife/SHA-analysis

Video Link at Youtube

https://youtu.be/6kQADPq5hic

Ucapan Terima Kasih

Penulis ingin memanjatkan rasa syukur kepada Tuhan Yang Maha Esa atas rahmat-Nya sehingga makalah ini dapat diselesaikan. Kemudian, penulis ingin mengucapkan terima kasih kepada orang tua dan keluarga yang selalu memberikan dukungan. Selain itu, penulis juga ingin mengucapkan terima kasih yang sebesar-besarnya atas segala arahan dan bimbingan yang diberikan oleh Dr. Ir. Rinaldi Munir, M.T. selaku dosen mata kuliah II4031 Kriptografi dan Koding STI.

References

[1] F. Mohammed, L. Nilesh, and P. Shalendra, “A Review Of Authentication Methods”, International Journal of Scientific & Technology Research, vol. 5, pp. 246-249, 2016.

[2] I. Velásquez, A. Caro, and A. Rodríguez, “Authentication schemes and methods: A systematic literature review,” Information and Software Technology, vol. 94, pp. 30–37, Feb. 2018, doi: https://doi.org/10.1016/j.infsof.2017.09.012.

[3] Shally, A. Singh, and G.A. Singh, “A Review Of One Time Password Mobile Verification”, International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR), vol. 4, pp. 113-118, 2014.

[4] L. Bošnjak, J. Sreš and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2018, pp. 1161-1166, doi: 10.23919/MIPRO.2018.8400211.

[5] M. Peyravian and N. Zunic, “Methods for Protecting Password Transmission,” Computers & Security, vol. 19, no. 5, pp. 466–469, Jul. 2000, doi: https://doi.org/10.1016/s0167-4048(00)05032-x.

[6] “How to use Hashcat for Ethical Hackers | Parrot OS,” www.youtube.com. https://www.youtube.com/watch?v=5fy6Lq1vgZk

[7] “how to HACK a password // password cracking with Kali Linux and HashCat,” www.youtube.com. https://www.youtube.com/watch?v=z4_oqTZJqCo

[8] “How to use Hashcat on Windows 10,” www.youtube.com. https://www.youtube.com/watch?v=rRUiRz_znLQ&t=358s

[9] Munir, R. Slide Kuliah Munir, R. Slide Kuliah Fungsi Hash

[10] Munir, R. Slide Kuliah Munir, R. Slide Kuliah Fungsi SHA-1

[11] Munir, R. Slide Kuliah Munir, R. Slide Kuliah Fungsi SHA-3

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, 22 Mei 2023

Siti Adira Ramadhani (18220094)