Simulasi Pemecahan Hash SHA-1 melalui Serangan Rainbow Table

Muhammad Dava Fathurrahman - 13522114[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522114@std.stei.itb.ac.id

Abstract*—Sistem komputer menyimpan kata sandi dengan menggunakan proses enkripsi hash untuk meningkatkan keamanan. Algoritma hash seperti SHA-1, meskipun digunakan luas sebelumnya, kini rentan terhadap serangan yang dapat memunculkan kesamaan nilai hash dari pesan yang berbeda. Serangan rainbow table, metode prakomputasi yang memanfaatkan kelemahan algoritma hash, telah menjadi ancaman serius terhadap keamanan kata sandi yang di-hash menggunakan SHA-1. Ketika terjadi kebocoran kata sandi melalui serangan rainbow table pada SHA-1, dampaknya bisa mengguncang keamanan sistem secara signifikan. Oleh karena itu, banyak organisasi beralih ke algoritma hash yang lebih kuat seperti SHA-256 yang menawarkan panjang hash yang lebih besar dan keamanan yang lebih solid. Kajian terhadap pembuatan tabel rainbow untuk fungsi hash SHA-1, menggunakan pustaka hashlib dan bahasa pemrograman Python, menunjukkan bahwa teknik ini mampu mengungkap kata sandi dalam situasi tertentu. Namun, studi lanjutan tentang kekuatan algoritma hash serta penggunaan metode keamanan yang lebih mutakhir perlu terus dilakukan guna menjaga keamanan sistem otentikasi yang luas digunakan.

***Keywords*— Enkripsi, Kriptografi, Keamanan, Kata Sandi, Hash, SHA-1**

I. PENDAHULUAN

Dalam sistem komputer, kata sandi tidak disimpan sebagai teks biasa, melainkan diolah menggunakan enkripsi hash. Fungsi hash merupakan proses satu arah, yang berarti tidak mungkin untuk mengembalikan nilai hash menjadi teks aslinya. Ketika seorang pengguna memasukkan kata sandi, sistem mengonversi kata sandi tersebut menjadi nilai hash yang kemudian dibandingkan dengan nilai hash yang sudah disimpan. Jika kedua nilai hash cocok, pengguna diautentikasi ke dalam sistem. Proses ini memastikan bahwa kata sandi tidak disimpan dalam bentuk teks biasa untuk keamanan lebih lanjut.

SHA-1 (Secure Hash Algorithm 1) merupakan salah satu algoritma hash yang paling banyak digunakan sebelum terungkap kerentanan keamanannya. Algoritma ini mengonversi pesan masukan menjadi hash 160 bit. Namun, seiring waktu, kelemahan keamanan dari SHA-1 terungkap, yang membuatnya rentan terhadap serangan yang dapat menemukan kolisi hash, yaitu dua pesan yang berbeda menghasilkan nilai hash yang sama.

Serangan *rainbow table* merupakan teknik yang memanfaatkan kelemahan algoritma hash, seperti SHA-1, dengan menggunakan tabel yang telah dihasilkan sebelumnya (prakomputasi) yang berisi sejumlah besar nilai hash beserta nilai asli yang menghasilkan hash tersebut. Dengan tabel ini, serangan dapat mencocokkan nilai hash yang ditargetkan dengan nilai-nilai hash yang telah dihitung sebelumnya dan disimpan dalam tabel. Dengan demikian, penyerang dapat dengan cepat menemukan nilai teks biasa (*plaintext*) yang sesuai dengan nilai hash yang telah ditemukan sebelumnya.

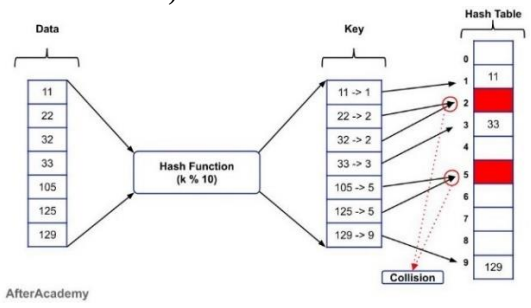
Ketika serangan *rainbow table* berhasil pada SHA-1, dampaknya bisa signifikan terhadap keamanan sistem. Inilah sebabnya mengapa banyak lembaga dan organisasi telah beralih ke algoritma hash yang lebih aman, seperti SHA-256 yang memiliki panjang hash yang lebih besar dan minim terjadinya kolisi.

II. LANDASAN TEORI

A. Fungsi Hash

Algoritma hash adalah metode kriptografi yang mengonversi data menjadi nilai hash yang unik. Hash seringkali berbentuk alfanumerik panjang dan merepresentasikan data yang lebih besar. Tujuan utamanya adalah untuk menghasilkan nilai hash yang unik untuk setiap input data yang berbeda sehingga sulit untuk menentukan nilai asli dari hash tersebut.

Fungsi hash harus memiliki beberapa sifat yang penting untuk keamanan dan kegunaan yang optimal. Pertama, mereka harus dapat menghasilkan nilai hash dengan cepat, bahkan untuk input yang besar. Kedua, perubahan sedikit pun pada input seharusnya menghasilkan perubahan nilai hash yang besar pula (*avalanche effect*). Ketiga, fungsi hash harus sulit untuk diinvers, artinya sulit untuk mengembalikan nilai hash ke input aslinya (*one-way property*). Terakhir, setiap nilai hash harus unik, sehingga tidak ada dua input yang berbeda yang menghasilkan nilai hash yang sama (*collision resistance*).



Gambar 1: Ilustrasi Hash

(Sumber: https://afteracademy.com/blog/the-concept-of-hashing-in-programming)

Pada gambar 1, suatu nilai integer k dilakukan hash dengan fungsi hashFunction(k) = k mod 10, ini akan menghasilkan nilai luaran antara 0 sampai dengan 9. Namun, algoritma hash ini kurang baik karena banyak integer k yang memiliki nilai hash yang sama (*collision*). Hal ini harus dihindari untuk fungsi hash dalam pengamanan data. Salah satu teknik untuk mengatasi kolisi adalah dengan menggunakan perantaian. Perantaian atau *chaining* adalah teknik resolusi kolisi di mana kita menggunakan daftar tertaut atau struktur data daftar untuk menyimpan kunci yang memiliki nilai yang sama.

Terdapat berbagai jenis hash functions yang umum digunakan dalam kriptografi modern. Beberapa di antaranya adalah MD5, SHA-1, SHA-256, dan SHA-512. Masing-masing memiliki panjang nilai hash yang berbeda dan tingkat keamanan yang bervariasi. Misalnya, SHA-1 (Secure Hash Algorithm 1) menghasilkan nilai hash sepanjang 160-bit berbentuk hexadecimal 40 digit. Pemilihan hash function yang tepat tergantung pada kebutuhan keamanan dan kecepatan komputasi yang diinginkan dalam suatu aplikasi tertentu.

B. Rainbow Table

Rainbow Table atau Tabel Pelangi adalah varian khusus dari tabel pencarian untuk membalikkan fungsi hash kriptografi, menggunakan mekanisme kompromi yang masuk akal antara waktu pencarian tabel dan jejak memori. Ide dasarnya adalah melakukan prakomputasi terhadap sejumlah besar nilai hash dan teks biasa yang mungkin. Data ini kemudian disimpan dalam tabel yang memetakan nilai hash ke teks biasa yang sesuai. Dalam serangan, tabel ini digunakan untuk mencocokkan nilai hash yang diperoleh dengan nilai hash dalam tabel, sehingga mengidentifikasi nilai *plaintext* yang cocok.

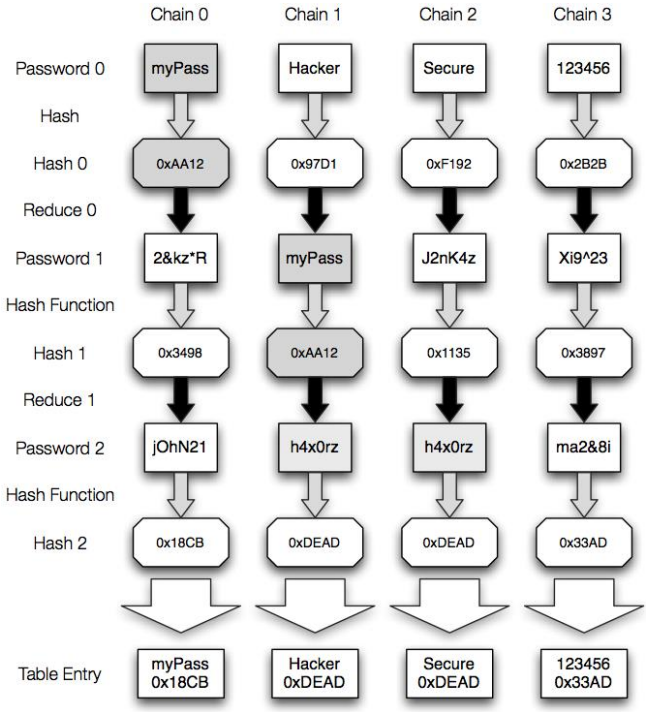
Chain atau rantai merupakan serangkaian nilai hash yang dihasilkan dari iterasi fungsi hash berulang. Dalam pembuatan tabel pelangi, rantai dimulai dengan titik awal yang merupakan nilai hash awal dari teks biasa yang telah dienkripsi menggunakan fungsi hash yang dipilih. Setiap nilai hash dalam rantai dihasilkan melalui iterasi menggunakan fungsi hash yang sama. Hasil dari iterasi ini menjadi nilai hash berikutnya dalam rantai, dan proses ini diulangi sejumlah kali untuk menciptakan rantai nilai hash yang panjang.

Fungsi reduksi adalah fungsi matematika yang memetakan kumpulan nilai hash kembali ke kumpulan teks biasa. Misalnya, suatu teks biasa P dengan panjang 6 karakter memiliki nilai hash dengan panjang 40 karakter. Dengan fungsi reduksi, kita dapat membentuk teks biasa lagi dengan cara mengubah 40 karakter ke 6 karakter. Hal ini dapat mengurangi jumlah nilai hash yang perlu disimpan dalam tabel. Dengan menggunakan fungsi reduksi yang tepat, jumlah nilai hash yang dihasilkan dapat diminimalkan, karena setiap nilai hash akan terhubung dengan nilai teks biasa yang mungkin melalui langkah reduksi.

Pembuatan tabel pelangi dimulai dengan memilih fungsi hash yang akan diserang, misalnya MD-5, SHA-1, dan SHA-256. Langkah berikutnya adalah menentukan nilai awal. Nilai-nilai awal ini dihasilkan dari teks biasa yang mungkin atau kandidat kata sandi melalui proses hash yang dipilih. Kemudian, nilai-nilai hash awal ini dijadikan sebagai titik awal untuk pembentukan rantai yang panjang. Setiap nilai hash dalam rantai dihasilkan melalui iterasi menggunakan fungsi hash yang sama, dan setiap langkah dalam rantai dihubungkan dengan langkah reduction yang akan mengonversi kembali nilai hash menjadi teks biasa yang mungkin. Proses iterasi dan reduksi ini dilakukan berulang hingga mencapai panjang rantai yang diinginkan.

Sebelum melakukan pembuatan tabel Pelangi, kita harus menentukan berbagai parameter, yaitu

1. Q adalah himpunan pembentuk password, misalnya alfabet, numerik, dan simbol, atau gabungan ketiganya.
2. N adalah panjang kata sandi atau kandidat password.
3. L adalah banyaknya rantai.
4. C adalah panjang rantai menyatakan banyaknya operasi hash.



Gambar 2: Rantai Hash

(Sumber: https://www.cryptohaze.com/wiki/index.php/Rainbow_Tables.html)

Gambar di atas menunjukkan pembuatan tabel dengan 4 rantai dengan 3 kali operasi hash. Langkah-langkah pemrosesan adalah sebagai berikut,

1. Kandidat password atau nilai awal diinisiasi pada baris “Password 0” sebagai nilai awal rantai.

2. Nilai awal di-hash dengan suatu fungsi hash, misalnya H(p), sehingga menghasilkan nilai pada baris “Hash 0”
3. Nilai pada “Hash 0” direduksi oleh suatu fungsi reduksi, misalnya $R_0(h)$, sehingga menghasilkan kandidat password baru pada baris “Password 1”.
4. Kandidat password baru ini di-hash lagi dengan H(p) sehingga menghasilkan nilai pada baris “Hash 1”
5. Nilai pada “Hash 1” direduksi oleh $R_1(h)$, sehingga menghasilkan kandidat password baru pada baris “Pasword 2”.
6. Kandidat password baru ini di-hash lagi dengan H(p) sehingga menghasilkan nilai pada baris “Hash 2”. Nilai pada baris “Hash 2” adalah nilai akhir rantai.
7. Pasangan nilai awal dan nilai akhir rantai disimpan pada tabel pelangi.

III. IMPLEMENTASI

Pada simulasi ini, penulis menggunakan konfigurasi khusus untuk melakukan serangan tabel pelangi. Dalam skenario ini, penulis memanfaatkan numerik sebagai himpunan pembentuk kata sandi, panjang kata sandi sebanyak 8 karakter, 5000 buah rantai dengan panjang setiap rantai sebesar 1000. Melalui simulasi ini, kita dapat mengamati proses pembangunan tabel rainbow serta penerapannya dalam upaya memecahkan nilai hash.

```python
import os
import hashlib
from random import randrange

charset = [i for i in range(10)]
plaintext_length = 8
num_hashes = 5000
chain_length = 1000
```

Gambar 3: Konfigurasi Pembangun Tabel Pelangi

```python
def hash_func(plaintext):
    return hashlib.sha1(plaintext.encode()).hexdigest()
```

Gambar 4: Fungsi Hash SHA-1

```python
def reduction_func(hash_string, iteration):
    value = (int(hash_string, 16) + iteration) % (2 ** 40)
    newPlaintext = ""
    for i in range(plaintext_length):
        idx = value % len(charset)
        newPlaintext += str(charset[idx])
        value //= len(charset)
    return newPlaintext
```

Gambar 5: Fungsi Reduksi

Fungsi “reduction_func” bertujuan untuk mengubah nilai hash SHA-1 berukuran 160-bit sedemikian rupa sehingga setiap kandidat kata sandi baru menjadi unik. Dengan setiap bit pada nilai hash SHA-1 memiliki dua kemungkinan nilai (0 atau 1), jumlah kombinasi yang mungkin mencapai 2^{160}. Penggunaan operasi modulo 2^{40} berusaha untuk mengurangi kemungkinan sebesar seperempatnya.

```python
def get_last_chain(hash_string, start_step = 0):
    newPlaintext = ""
    for i in range(start_step, chain_length):
        newPlaintext = reduction_func(hash_string, i)
        hash_string = hash_func(newPlaintext)
    return newPlaintext
```

Gambar 6: Fungsi Nilai Akhir Rantai

Fungsi "get_last_chain" merupakan elemen kunci dalam pembentukan tabel pelangi untuk serangan terhadap fungsi hash. Fungsi ini bertanggung jawab untuk membentuk serangkaian langkah reduksi dari nilai hash awal. Pada setiap langkah, nilai hash saat ini diubah menjadi *plaintext* baru melalui fungsi reduksi. *Plaintext* yang dihasilkan kemudian diolah kembali menggunakan fungsi hash, membentuk suatu rangkaian langkah yang berlanjut hingga mencapai langkah terakhir dalam serangkaian reduksi. Hasil akhir dari fungsi ini adalah *plaintext* yang dihasilkan setelah melalui seluruh langkah dalam rantai reduksi. Selain menghasilkan *plaintext* baru, fungsi ini juga efisien dalam penggunaan ruang data karena proses rantai memungkinkan penghematan space yang signifikan dalam penyimpanan data tabel pelangi yang digunakan dalam serangan terhadap fungsi hash.

```python
def generate():
    for i in range(num_hashes):
        try:
            start_word = ""
            start_word += ''.join([str(charset[randrange(len(charset))])\
                                   for _ in range(plaintext_length)])
            end_word = get_last_chain(hash_func(start_word))

            with open('table.txt', 'a') as file:
                file.write(f"{start_word}:{end_word}\n")
        except StopIteration:
            break
```

Gambar 7: Fungsi Pembuat Tabel Pelangi

Fungsi "generate" digunakan dalam pembentukan dan penyimpanan data pada tabel pelangi yang akan digunakan dalam serangan terhadap fungsi hash. Melalui iterasi sejumlah "num_hashes" yang telah ditentukan, fungsi ini membentuk sejumlah entri pada tabel pelangi. Setiap entri tabel dibangun dengan cara membangkitkan kata awal secara acak dengan panjang yang telah ditentukan oleh "plaintext_length". Langkah berikutnya melibatkan penggunaan fungsi "get_last_chain" untuk menghasilkan nilai akhir dari rantai reduksi berdasarkan kata awal yang telah dibuat. Hasil akhir dari rantai ini kemudian disimpan sebagai pasangan "start_word" dan "end_word" dalam sebuah fail teks dengan nama "table.txt".

```python
def get_from_table(end_word):
    try:
        result = []
        with open('table.txt', 'r') as file:
            for line in file:
                start_word, current_end_word = line.strip().split(':')
                if current_end_word == end_word:
                    result.append(start_word)
        return result
    except Exception as exception:
        print("Error reading from file:", exception)
        return []
```

Gambar 8: Fungsi Pencarian pada Tabel Pelangi

Fungsi "get_from_table" berperan dalam mencari nilai awal (start_word) berdasarkan nilai akhir (end_word) yang diberikan pada tabel pelangi yang telah dibuat sebelumnya. Melalui proses ini, fungsi membuka file teks dengan nama "table.txt" yang berisi pasangan nilai awal dan nilai akhir dari setiap rantai reduksi. Dengan membaca fail baris per baris, fungsi ini memeriksa setiap entri dalam tabel untuk menemukan nilai awal yang sesuai dengan nilai akhir yang dicari. Ketika nilai akhir cocok dengan nilai akhir yang diberikan, fungsi menyimpan nilai awal tersebut dalam sebuah daftar (*list*) dan mengembalikan hasil tersebut sebagai keluaran dari fungsi ini.

```python
def get_password(hash_string):
    for i in range(chain_length - 1, -1, -1):
        end_word = get_last_chain(hash_string, i)
        result = get_from_table(end_word)
        if len(result) > 0:
            for word in result:
                for step in range(0, chain_length):
                    current_hash = hash_func(word)
                    if hash_string == current_hash:
                        return word
                    word = reduction_func(current_hash, step)
```

Gambar 9: Fungsi Pencarian Kata Sandi

Fungsi "get_password" berguna untuk menemukan nilai awal (plaintext) yang sesuai dengan nilai hash yang diberikan. Fungsi ini melakukan serangkaian langkah mundur (*reverse lookup*) terhadap rantai reduksi dengan memanfaatkan informasi yang telah disimpan pada tabel pelangi sebelumnya. Dengan memanfaatkan nilai hash awal yang diberikan, fungsi ini menggunakan serangkaian langkah mundur untuk menghasilkan nilai akhir dari rantai reduksi pada setiap langkahnya. Kemudian, fungsi "get_from_table" digunakan untuk mencari nilai awal yang cocok dengan nilai akhir yang dihasilkan tersebut pada tabel pelangi. Jika ada nilai awal yang sesuai ditemukan, fungsi akan memeriksa setiap langkah pada rantai reduksi ini dengan menghitung ulang nilai hash pada setiap langkah reduksi. Ketika nilai hash yang dihasilkan sama dengan nilai hash yang diberikan, maka nilai awal yang sesuai telah ditemukan, dan nilai awal ini akan dikembalikan sebagai kata sandi yang sesuai dengan nilai hash yang diberikan.

```python
def main():
    if not os.path.exists("table.txt"):
        generate()

    print("""
 ____       _       _                      _____     _     _           _   _   _             _
|  _ \ __ _(_)_ __ | |__   _____      __  |_   _|_ _| |__ | | ___     / \ | |_| |_ __ _  ___| | _____ _ __
| |_) / _` | | '_ \| '_ \ / _ \ \ /\ / /    | |/ _` | '_ \| |/ _ \   / _ \| __| __/ _` |/ __| |/ / _ \ '__|
|  _ < (_| | | | | | |_) | (_) \ V  V /     | | (_| | |_) | |  __/  / ___ \ |_| || (_| | (__|   <  __/ |
|_| \_\__,_|_|_| |_|_.__/ \___/ \_/\_/      |_|\__,_|_.__/|_|\___| /_/   \_\__|\__\__,_|\___|_|\_\___|_|
    """)
    password_hash = input("Input SHA-1 Hash: ")
    found_password = get_password(password_hash)

    if found_password:
        print("Plaintext is:", found_password)
    else:
        print('Sorry, no results found!')


if __name__ == "__main__":
    main()
```

Gambar 10: Fungsi Utama dalam Penggunaan Program

```
$ python rainbow-table-attacker.py

 ____       _       _                      _____     _     _           _   _   _             _
|  _ \ __ _(_)_ __ | |__   _____      __  |_   _|_ _| |__ | | ___     / \ | |_| |_ __ _  ___| | _____ _ __
| |_) / _` | | '_ \| '_ \ / _ \ \ /\ / /    | |/ _` | '_ \| |/ _ \   / _ \| __| __/ _` |/ __| |/ / _ \ '__|
|  _ < (_| | | | | | |_) | (_) \ V  V /     | | (_| | |_) | |  __/  / ___ \ |_| || (_| | (__|   <  __/ |
|_| \_\__,_|_|_| |_|_.__/ \___/ \_/\_/      |_|\__,_|_.__/|_|\___| /_/   \_\__|\__\__,_|\___|_|\_\___|_|

Input SHA-1 Hash: 70352f41061eda4ff3c322094af068ba70c3b38b
Plaintext is: 00000000
```

Gambar 11: Eksperimen Pemecahan SHA-1 (1)

```
$ python rainbow-table-attacker.py

 ____       _       _                      _____     _     _           _   _   _             _
|  _ \ __ _(_)_ __ | |__   _____      __  |_   _|_ _| |__ | | ___     / \ | |_| |_ __ _  ___| | _____ _ __
| |_) / _` | | '_ \| '_ \ / _ \ \ /\ / /    | |/ _` | '_ \| |/ _ \   / _ \| __| __/ _` |/ __| |/ / _ \ '__|
|  _ < (_| | | | | | |_) | (_) \ V  V /     | | (_| | |_) | |  __/  / ___ \ |_| || (_| | (__|   <  __/ |
|_| \_\__,_|_|_| |_|_.__/ \___/ \_/\_/      |_|\__,_|_.__/|_|\___| /_/   \_\__|\__\__,_|\___|_|\_\___|_|

Input SHA-1 Hash: 435fc4d8aa8774b0456da37d3a71d01f72c742e0
Plaintext is: 87263549
```

Gambar 12: Eksperimen Pemecahan SHA-1 (2)

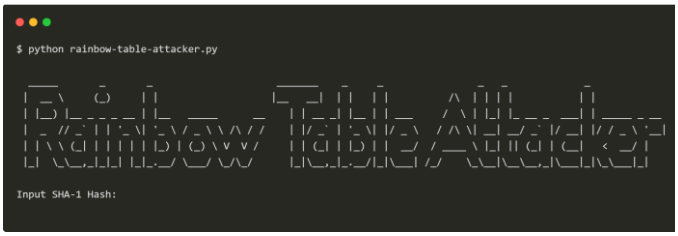
```
$ python rainbow-table-attacker.py

 ____       _       _                      _____     _     _           _   _   _             _
|  _ \ __ _(_)_ __ | |__   _____      __  |_   _|_ _| |__ | | ___     / \ | |_| |_ __ _  ___| | _____ _ __
| |_) / _` | | '_ \| '_ \ / _ \ \ /\ / /    | |/ _` | '_ \| |/ _ \   / _ \| __| __/ _` |/ __| |/ / _ \ '__|
|  _ < (_| | | | | | |_) | (_) \ V  V /     | | (_| | |_) | |  __/  / ___ \ |_| || (_| | (__|   <  __/ |
|_| \_\__,_|_|_| |_|_.__/ \___/ \_/\_/      |_|\__,_|_.__/|_|\___| /_/   \_\__|\__\__,_|\___|_|\_\___|_|

Input SHA-1 Hash: fe53e6cb894cd455ef24240f9824f3ec52ea0bb1
Sorry, no results found!
```

Gambar 13: Eksperimen Pemecahan SHA-1 (3)

Melalui ketiga eksperimen yang terdokumentasi dalam Gambar 11, Gambar 12, dan Gambar 13, dipertunjukkan bagaimana serangan rainbow mampu menemukan kesesuaian antara nilai hash yang diperoleh dari sistem yang diserang dengan plaintext yang sesuai dalam tabel rainbow yang telah diprakomputasi.

IV. HASIL DAN PEMBAHASAN

Saat program dijalankan, pertama-tama, program akan mengecek keberadaan “table.txt” yang merupakan rainbow table. Jika tabel belum tersedia, program akan memanggil fungsi “generate” untuk membuat rainbow table. Selanjutnya, pengguna diminta untuk memasukkan Hash SHA-1 yang valid dan ingin didekripsi.



Gambar 14: Program Menunggu Masukan dari Pengguna

Setelah pengguna memasukkan nilai hash SHA-1, program akan memanggil fungsi “get_password” untuk melakukan pencarian plainteks dari tabel pelangi. Jika plaintext ditemukan, program akan menampilkan pesan “Plaintext is: “ dan diikuti dengan plainteks yang ditemukan. Jika plaintext tidak ditemukan, program akan menampilkan pesan “Sorry, no results found!”.

Penulis melakukan pengujian program dengan mempertimbangkan tiga kasus yang berbeda. Pertama, plaintext tersedia pada tabel pelangi dan menjadi nilai awal (start_word). Kedua, plaintext juga ada dalam tabel pelangi, namun posisinya terletak di antara langkah-langkah dalam proses reduksi. Ketiga, dalam kasus terakhir, plaintext tidak terdaftar dalam tabel pelangi.

Dalam kasus pertama, pasangan 86532678:15461677 terdapat dalam tabel pelangi. Nilai hash dari 86532678 dalam algoritma SHA-1 adalah 012a402cc71d1f210e3418e75f7cace7fce2813a. Berdasarkan data yang terdokumentasi dalam Gambar 15, program berhasil mengidentifikasi plaintext dari nilai hash SHA-1 yang diuji.

Gambar 15: Hasil Eksperimen Kasus I

Sementara dalam kasus kedua, plaintext 01234567 dipastikan bukan bagian dari pasangan nilai awal dan akhir dalam tabel pelangi. Hash dari 01234567 dalam SHA-1 adalah ccaa8d8dcc7d030cd6a6768db81f90d0ef976c3d. Berdasarkan informasi yang tercatat dalam Gambar 16, program berhasil menemukan plaintext dari nilai hash SHA-1 yang diuji. Hal ini mengindikasikan bahwa tabel pelangi efektif dalam menjalankan proses reduksi, mampu menemukan plaintext baru yang terletak di antara langkah-langkah dalam proses reduksi dan sekaligus menghemat ruang penyimpanan.

Gambar 16:Hasil Eksperimen Kasus II

Pada kasus terakhir, plaintext dipastika tidak terdaftar dalam tabel pelangi milik penulis, misalnya 99998888. Nilai hash dari 99998888 dalam algoritma SHA-1 adalah 545dcfbb5bb3a04a7acefc6ca2f03e517e2a0025. Berdasarkan data yang terdokumentasi dalam Gambar 17, program tidak berhasil mengidentifikasi plaintext dari nilai hash SHA-1 yang diuji.

Gambar 17:Hasil Eksperimen Kasus III

Secara keseluruhan, program belum dapat menemukan seluruh plaintext dari masukan SHA-1 pengguna. Hal ini disebabkan oleh tabel pelangi dengan metode pembentukan nilai awal plaintext yang didasarkan pada fungsi acak, bukan dari permutasi seluruh kemungkinan kata sandi yang lebih sistematis.

V. KESIMPULAN DAN SARAN

Penelitian tentang ketahanan algoritma hash menjadi semakin penting dalam sistem otentikasi yang digunakan secara luas. Algoritma hashing yang umum perlu dipertimbangkan untuk diperbarui dan dievaluasi lebih lanjut untuk menjamin keamanannya dalam menghadapi tantangan keamanan modern. Serangan menggunakan tabel pelangi, yang merupakan metode pra-perhitungan untuk memecahkan kata sandi yang di-hash, telah terbukti efektif terhadap kata sandi yang sederhana. Studi yang dilakukan terhadap pembuatan tabel pelangi untuk fungsi hash SHA-1, menggunakan pustaka hashlib dan bahasa pemrograman Python, menunjukkan kemampuan tabel pelangi dalam mengungkapkan kata sandi yang digunakan dalam situasi tertentu.

Penulis menyarankan penggunaan metode permutasi yang terstruktur untuk nilai awal pada tabel pelangi. Dengan

mengadopsi pendekatan ini, diharapkan program mampu mengatasi batasan yang disebabkan oleh penggunaan nilai awal secara acak. Selain itu, mendesain dan menerapkan fungsi reduksi yang lebih efektif akan menjadi kunci untuk meminimalisir kemungkinan kolisi dalam proses pembentukan tabel pelangi. Dengan fungsi reduksi yang optimal, dapat diantisipasi bahwa program akan mampu menangani lebih banyak kombinasi nilai hash dengan lebih baik dan meningkatkan keberhasilan dalam menemukan plaintext yang sesuai.

VI. UCAPAN TERIMA KASIH

Penulis dengan tulus bersyukur kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya yang telah memungkinkan penyelesaian makalah ini tepat waktu. Tidak lupa juga rasa terima kasih yang mendalam kepada Ibu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T., selaku dosen pengampu mata kuliah IF2120 Matematika Diskrit kelas K02, yang telah memberikan bimbingan yang sangat berharga, serta pengetahuan yang melimpah dalam penulisan makalah ini. Kehadiran dan arahan beliau telah sangat membantu dalam memahami materi secara lebih mendalam serta memberikan pandangan yang sangat berarti terhadap isi makalah ini. Penulis juga mengucapkan terima kasih kepada orang tua atas dukungan moral dan spiritual yang telah diberikan, serta inspirasi dan motivasi yang senantiasa mereka berikan selama proses penulisan makalah.

REFERENSI

[1] Manankova, O.A., Yakubova, M.Z., Rakhmatullaev, M.A., & Baikenov, A.S. (2023). Simulation of the Rainbow Attack on the SHA-256 Hash Function. Journal of Theoretical and Applied Information Technology.

[2] Admin AfterAcademy. (2020, January 15). The Concept of Hashing in Programming. Diakses dari https://afteracademy.com/blog/the-concept-of-hashing-in-programming pada 10 Desember 2023.

[3] Andreev, D. (2021, March 2). Rainbow Table. [GitHub Gist]. Diakses dari https://gist.github.com/DanilAndreev/77036b5f7a7dc656c54aacb31140c22c (Diakses pada 7 Desember 2023).

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/16-Teori-Bilangan-Bagian3-2023.pdf (Diakses pada 7 Desember 2023).

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, 11 Desember 2023

Muhammad Dava Fathurrahman
13522114