Implementasi Teori Bilangan dan Graf pada Algoritma Pembuatan *One-Time Password* (OTP) untuk Peningkatan Keamanan

Diana Tri Handayani - 13522104[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522104@std.stei.itb.ac.id

***Abstract*—Algoritma pembangkitan *One-Time Password* pada umumnya berdasarkan HMAC atau berdasarkan waktu. Namun, seiring berjalannya waktu, tingkat kemampuan peretasan semakin maju. Oleh karen itu diperlukan inovasi dalam pembuatan algoritma pembangkitan kode OTP. Salah satunya dengan memanfaatkan graf lengkap yang dapat membuat kombinasi sirkuit Hamilton yang cukup banyak. Kemungkinan tersebut hingga mencapai (n-1)!/2 dengan n sadalah banyak simpul graf lengkap. Dengan adanya tambahan kemungkinan kombinasi algoritma diharapkan tingkat keamanan algoritma pembangkitan OTP ini semakin meningkat dan tercegah dari adanya peretasan yang tidak diinginkan.**

Keywords—graf, Hamilton, hash, One-Time Password.

I. PENDAHULUAN

Saat ini, sudah banyak kegiatan yang mulai didigitalisasi, bahkan untuk kegiatan-kegiatan yang krusial seperti mengirimkan uang antar individu. Tentu saja, digitalisasi dilakukan untuk mendapatkan kemudahan dan efisiensi. Tetapi, dengan didapatkan kemudahan tersebut, munculah risiko kejahatan yang sama mudahnya untuk terjadi. Oleh karena itu, diperlukannya keamanan siber dan informasi guna mencegah adanya penyalahgunaan yang tidak diinginkan.

Salah satu cara untuk mencegah adanya penyalahgunaan adalah dengan menggunakan otentikasi pengguna. Otentikasi pengguna dapat menggunakan berbagai cara, salah satunya yang sudah banyak digunakan masyarakat ialah *One-Time Password* (OTP). Konsep dasar dari OTP adalah memberikan kode sandi yang hanya berlaku untuk satu kali penggunaan atau untuk suatu sesi tertentu. Hal ini mengatasi kerentanan keamanan yang muncul dari penggunaan kata sandi yang statis, yang dapat dengan mudah dicuri atau diretas. Dengan demikian, OTP memberikan keamanan tambahan dengan memastikan bahwa setiap kali pengguna mengakses sistem atau melakukan transaksi, kode atau *password* yang digunakan akan berbeda.

Kategori *One-Time Password* yang telah banyak digunakan secara umum ialah *Time-based* OTP (TOTP) yang merupakan perkembangan dari HMAC-*based* OTP (HOTP). Maksud dari *Time-based* adalah masukan fungsi hash tidak hanya bergantung pada kata kunci tetapi juga pada waktu secara '*real-time'* sehingga dapat meningkatkan tingkat keamanan.

Tingkat keamanan pembentukan *password* tersebut bergantung pada fungsi hash yang membentuk kode tersebut. Singkatnya telah terdapat *Secure Hash Algorithm (SHA)* yang telah dipercayai cukup aman untuk menghindari adanya penyerangan terhadap *password* yang dibuat. Namun, tidak dipungkiri perkembangan kemampuan peretasan juga semakin pesat, sehingga dibutuhkan tingkat keamanan yang lebih lagi. Oleh karena itu, penulis akan mencoba memodifikasi algoritma pembuatan OTP dengan menggunakan konsep teori bilangan dan graf (khususnya graf Hamilton) untuk meningkatkan tingkat keamanan dari serangan peretasan.

II. LANDASAN TEORI

A. *Fungsi Hash*

Fungsi Hash ialah salah satu implementasi dari teori bilangan yang digunakan untuk pengalamatan data di dalam memori agar data dapat diakses dengan cepat. Fungsi hash akan melakuakan kompresi sebuah pesan (M) yang berukuran sembarang menjadi sebuah string (h) yang ukurannya bersifat pasti (*fixed*). Ukuran string (h) secara umum akan menghasilkan ukuran yang jauh lebih kecil daripada ukuran pesan (M) sebelumnya. Fungsi hash ini bersifat tak-dapat kembali dengan kata lain fungsi ini hanya satu-arah (*one-way hash* function). Bentuk umum fungsi hash dapat dituliskan sebagai berikut:

$$h = H(M)$$; dengan,

h = hash value = message digest = digest;
H = fungsi hash;
M = pesan yang akan dikonversi.

Dalam fungsi hash terdapat istilah kolisi (*collision*) yang mengartikan terdapat dua string sembarang memiliki nilai hash yang sama. Kolisi tersebut menunjukkan fungsi hash tidak aman secara kriptografis. Sehingga fungsi hash yang valid, untuk setiap x yang diberikan, tidak mungkin didapatkan y dan x sedemikian sehingga $H(y) = H(x)$.

B. *Secure Hash Algorithm (SHA)*

Secure Hash Algorithm (SHA), sesuai namanya, merupakan fungsi hash satu-arah yang dikembangkan oleh *National Institute of Standards and Technology* (NIST). SHA telah banyak terjadi perkembangan sejak dikeluarkan pada tahun

1993. Mulai dari SHA-0, SHA-1, SHA-2 hingga SHA-3. Perkembangan atau revolusi tersebut secari garis besar untuk meningkatan keamanan dengan memperpanjang bit sehingga kemungkinan terjadi kolisi akan semakin kecil.

Pada kali ini, hanya akan fokus untuk membahas SHA-256 yang akan digunakan ketika pengimplementasian algoritma modifikasi nanti. Algoritma SHA-256 merupakan algoritma hash dari jenis SHA-2 yang menghasilkan nilai hash sebesar 256 bit. Sebelum dilakukan Algoritma SHA-256, perlu adanya penyesuain input dengan mengubah pesan ke dalam bentuk biner. Selanjutnya dilakukan penambahan *padding* bit hingga Panjang pesan kongruen dengan 448 (mod 512). Lalu, terdapat penambahan Panjang pesan sebanyak 64 bit di akhir.

Tahap selanjutnya melakukan parsing pesan dengan membagi setiap blok 512 bit menjadi 16 blok dengan masing-masing berukuran 32 bit. Barulah dilakukan inisiasi nilai hash dan diperoleh nilai K yang berasal dari konstanta SHA-256. Langkah selanjutnya adalah penjadwalan pesan, yatiu diawali denga mengubah stiap blok pesan menjadi bilangan heksadesimal dengan ketentuan sebagai berikut:

$$Wt \begin{cases} Mt^{(t)} & 0 \le t \le 15 \\ \sigma_1^{(256)}(W_{t-2}) + (W_{t-7}) + \sigma_0^{(256)}(W_{t-15}) + (W_{t-16}) & 16 \le t \le 63 \end{cases} \quad (1)$$

Di mana :

$\sigma 1(256)(Wi{-}2) = ((Wi{-}2)ROTR\ 17) \oplus ((Wi{-}2)ROTR\ 19) \oplus ((Wi{-}2)SHR10)$

$\sigma 1(256)(Wi{-}15) = ((Wi{-}15)ROTR\ 7) \oplus ((Wi{-}15)ROTR\ 18) \oplus ((Wi{-}15)SHR3)$

Keterangan :

- Wt = Blok Pesan yang Baru
- Mt = Blok Pesan yang Lama
- Wi-2 = Blok Pesan dari W ke i-2
- Wi-15 = Blok Pesan dari W ke i-15
- ROT R = *Rotate Right*
- SH R = *Shift Right*
- ⊕ = Operator XOR

Sumber: JURNAL MEDIA INFORMATIKA BUDIDARMA (stmik-budidarma.ac.id).

Terakhir akan dilakukan penjumlahan hasil akhir dan didapatkan keluaran yang menggabungkan dari hasil semua *hashing*.

C. *Time-based One-Time Password*

One-Time Password (OTP) adalah kata sandi yang valid atau dapat digunakan hanya untuk satu kali sesi. OTP biasanya digunakan sebagai otentikasi tambahan atau sering disebut juga sebagai dua faktor otentikasi (*second factor authentication-SFA*). Mekanisme otentikasi sendiri bertujuan untuk membuktikan bahwa pengguna yang sedang menggunakan sistem adalah orang yang memang memiliki otoritas.

Peran OTP yang cukup penting, membuat algoritma yang membangkitkan kata sandi tersebut harus benar-benar dipastikan tingkat keamanannya. Namun, juga harus memperhatikan tingkat kemudahan penggunaan dari sisi pengguna. Oleh karena itu, diadakannya OTP yang berdasarkan waktu saat digunakan. Biasanya OTP hanya berupa 6 digit angka yang di-*generate* setiap akan digunakan dan hangus dalam suatu kurun waktu. Sehingga, tingkat kemudahan dan tingkat keamanan dapat diperoleh secara sekaligus.

Algoritma Time-based OTP (TOTP) merupakan perkembangan dari bentuk HMAC-based OTP (HOTP). HMAC adalah salah satu tipe dari *Message Authentication Code (MAC)* yang memiliki basis fungsi hash satu-arah. Bentuk umum HOTP adalah:

HOTP(K, C) = truncate(HMAC-SHA-1 (K, C).

Sedangkan TOTP yang merupakan bentuk perkembangannya memiliki bentuk umum:

TOTP = HOTP (K, T)

dengan K adalah kata kunci dan T adalah nilai integer yang merepresentasikan waktu dengan diperoleh dari :

$$T = \frac{Tcurrent - T0}{x}$$

Dengan Tcurrent adalah waktu saat ini dalam detik dan T0 adalah nilai kesepakatan ketika awal inisialisasi. Sedangkan x adalah parameter untuk menentukan kurun waktu kevalidan kode OTP.

D. *Graf*

Didefinisikan graf G = (V, E) dengan V adalah himpunan tak-kosong yang berisi simpul-simpul dan E adalah himpunan sisi-sisi yang menghubungkan simpul. Terdapat banyak jenis graf berdasarkan bentuk fisik amupun sifatnya. Salah satunya adalah graf Hamilton.

Graf Hamilton merupakan graf yang memiliki sirkuit hamilton. Sirkuit Hamilton adalah sirkut yang melalui tiap simpul di dalam graf tepat satu kali, kecuali simpul asal yang harus sekaligus simpul akhir (dilalui dua kali). Sedangkan untuk graf yang hanya memiliki lintasan Hamilton (bukan sirkuit) disebut graf semi-Hamilton.

Terdapat teorema yang menyatakan bahwa setiap graf lengkap adalah graf Hamilton. Di dalam graf lengkap G dengan n buah simpul dan n lebih dari sama dengan 3, maka terdapat (n-1)!/2 buah sirkut Hamilton yang dapat terbentuk. Graf Hamilton ini telah banyak digunakan untuk mengatasi masalah aktual seperti persoalan pedagang keliling (*Travelling Salesperson Problem- TSP*) yang harus menyinggahi setiap kota tepat satu kali dan kembali ke kota asal. Namun, untuk kali ini graf Hamilton akan dimanfaatkan melalui banyaknya kemungkinan sirkut Hamilton yang dapat dibentuk dalam suatu graf lengkap.

III. IMPLEMENTASI

A. *Algoritma Sederhana Time-based OTP*

Berdasarkan beberapa referensi dari algoritma yang sudah ada, penulis membuat ulang dalam bentuk sederhana untuk mengetahui garis besar jalan algoritma. Didapatkan algoritma dalam bahasa *python* sebagai berikut:

- Algoritma ini memanfaatkan *library* dari python untuk mempermudah pembangunan algoritma. Pertama, *library* hashlib yang menyediakan implementasi dari berbagai fungsi hash kriptografis, salah satunya SHA-256 yang akan digunakana. Fungsi ini berguna untuk mengamankan dan mengintegrasikan data. Kedua *library* hmac digunakan untuk menghasilkan HMAC, yaitu kode otentikasi pesan berbasis hash. Ketiga, *library* struct digunakan untuk mengonversi antara representasi biner dan representasi terstruktur data. Serta *library* time menyediakan fungsi-fungsi time yang akan digunakan untuk mengambil data waktu terkini sebagai masukan fungsi hash nantinya.

```
import hashlib
import hmac
import struct
import time
```

- Terdapat fungsi generate_totp_sha256 untuk mendapatkan kode otp yang berdasarkan kata kunci dan waktu. Keterangan lebih lanjut terdapat pada gambar (dalam bentuk komentar).

```
def generate_totp_sha256(secret, digits, time_step=5):
    # Parameters:
    # secret (bytes) sbg kata kunci,
    # time_step (int) dalam satuan detik,
    # digits (int) banyaknya digit otp yang akan dihasilkan

    # Hitung time step terkini
    current_time_step = int(time.time() / time_step)

    # Ubah time step menjadi byte array (big-endian)
    time_step_bytes = struct.pack(">Q", current_time_step)

    # Hitung nilai HMAC menggunakan SHA-256
    hmac_digest = hmac.new(secret, time_step_bytes, hashlib.sha256).dig

    # Dynamic Truncation, ekstraksi 4 bytes dari nilai HMAC
    offset = hmac_digest[-1] & 0x0F
    truncated_hash = hmac_digest[offset : offset + 4]

    # Ubah hasil menjadi bentuk integer
    otp_value = struct.unpack(">I", truncated_hash)[0]

    # Ambil sebanyak digit yang diinginkan
    totp = otp_value % 10 ** digits

    # Formatting jika angka awal berupa '0'
    return f"{totp:0{digits}}"
```

Dari fungsi tersebut akan didapatkan 6 digit OTP yang berasal dari *hashing* kata kunci dan data waktu.

B. Algoritma Pembangkitan OTP dengan Graf

Penulis telah mencoba membuat algoritma untuk mendapatkan 6 digit OTP yang berasal dari sirkut Hamilton graf lengkap. Berikut algoritma yang telah dibuat:

- Algoritma ini memanfaatkan *library* dari python untuk mempermudah pembangunan algoritma.

```
import networkx as nx
import random
```

Library networkx merupakan library python yang dgiunakan untuk memodelkan, menganalisis, dan memvisualisasikan struktur jaringan (graf). Ini menyediakan kelas-kelas dan fungsi-fungsi untuk bekerja dengan berbagai jenis graf, termasuk graf berarah, graf tidak berarah, graf berbobot, dan sebagainya. Sehingga ini akan sangat berguna ketika pembuatan graf yang diperlukan dalam pembangkitan kode OTP bebrbasi graf. Sedangkan *library* random menyediakan fungsi-fungsi untuk menghasilkan angka-angka acak. Hal ini berguna ketika akan menentukan bobot tiap sisi dalam graf lengkap yang cukup banyak, hanya dengan random maka akan teracak secara otomatis.

- Sebelum masuk ke algoritma fungsi utama pembuatan graf, diperlukan fungsi-fungsi pembantu. Pertama adalah fungsi generated_weighted_complete_graph yang digunakan untuk membuat graf lengkap berbobot dengan masukan jumlah simpul dan rentang bobot.

```
def generate_weighted_complete_graph(num_nodes, min_weight, max_weight):
    G = nx.complete_graph(num_nodes)

    # Generate bobot pada tiap sisi yang ada menggunaakn library random
    weighted_edges = [(u, v, random.randint(min_weight, max_weight)) for u, v in G.edges]
    G.add_weighted_edges_from(weighted_edges)
    return G
```

Kedua adalah fungsi is_hamilton_path untuk mengecek apakah sebuat jalur masukan fungsi berupa sirkuit Hamilton.

```
def is_hamiltonian_path(graph, path):
    # Menguji apakah sebuah jalur masukan 'path' berupa hamilton,
    # dengan memastikan simpul yang dilewati tepat sastu kali menggunakan fungsi set,
    # serta memerikasa setiap simpul u dan v terhubung di graph
    return set(path) == set(graph.nodes) and all(u in graph[v] for u, v in zip(path, path[1:]))
```

Ketiga yaitu fungsi find_random_hamilton_path untuk mencari secara random sirkuit Hamilton.

```
def find_random_hamiltonian_path(graph):
    # Membuat daftar simpul dalam graf
    nodes = list(graph.nodes)

    # Mengacak urutan simpul
    random.shuffle(nodes)

    # Iterasi melalui semua permutasi simpul
    for path in [nodes[i:] + nodes[:i] for i in range(len(nodes))]:
        # Memeriksa apakah jalur yang dihasilkan memenuhi kriteria jalur Hamiltonian
        if is_hamiltonian_path(graph, path):
            return path  # Jalur Hamiltonian ditemukan
    return None  # Tidak ada jalur Hamiltonian dalam graf
```

Keempat, fungsi list_to_integer untuk mengubah sebuah list yang berisi beberapa integer menjadi satu integer saja dengan cara menggabungkannya.

```
def list_to_integer(digits):
    # Pastikan digits berisi angka-angka
    if not all(isinstance(digit, int) for digit in digits):
        raise ValueError("Semua elemen dalam list harus berupa integer.")

    # Menggabungkan digit menjadi satu bilangan integer
    result = int("".join(map(str, digits)))
    return result
```

- Fungsi utama pembentukan kode OTP dengan sirkuit Hamilton.

```
def generate_hamilton_code(num_nodes, min_weight, max_weight):
    # Generate graph dari masukan fungsi
    weighted_graph = generate_weighted_complete_graph(num_nodes, min_weight, max_weight)

    # Mencari jalur hamilton
    hamiltonian_path = find_random_hamiltonian_path(weighted_graph)

    # Mengubah hasil hamilton yang dalam bentuk list menjadi integer 6 digit
    hamilton_10digits = list_to_integer(hamiltonian_path)
    hamilton = hamilton_10digits % 10 ** 6

    # Mencari total bobot sisi yang dilalui siklus hamilton
    total_weight = 0
    if hamiltonian_path:
        for i in range(len(hamiltonian_path)):
            node = hamiltonian_path[i]
            next_node = hamiltonian_path[(i + 1) % len(hamiltonian_path)]
            edge_weight = weighted_graph[int(node)][int(next_node)]['weight']
            total_weight += edge_weight
    else:
        print("Tidak ada jalur Hamilton yang ditemukan.")
    return f"{hamilton:0{digits}}", total_weight
```

Fungsi ini selain mengembalikan kode OTP 6 digit, juga mengembalikan total bobot dari sisi yang dilalui dalam pembuatan sirkuit Hamilton.

C. Algoritma Utama

Dalam algoritma utama dilakukan kombinasi antara kode OTP dari Time-based OTP dan sirkuit Hamilton. Kombinasi dilakukan dengan menggunakan fungsi combine_digits dengan

konsep memanfaatkan jumlah bobot sisi sirkut hamilton di modulus dengan 6 sebagai banyak digit OTP.

```python
def combine_digits(number1, number2, modulus_value):
    # Ambil digit pertama dari number1 sebanyak hasil modulus dan sisa digit terakhir dari number2
    result = int(str(number1)[:modulus_value] + str(number2)[modulus_value:])
    return result
```

Masuk ke program utama untuk memanggil fungsi pembangkit OTP baik berdasarkan waktu maupun sirkuit Hamilton, serta menggabungkannya.

```python
if __name__ == "__main__":
    # Banyak digit yang akan dihasilkan
    digits = 6

    # Secret key, harus dirahasiakan
    secret_key = b"MySecretKey123"

    # Generate TOTP menggunakan SHA-256
    totp = generate_totp_sha256(secret_key, digits)

    # Definisikan banyak nodes dan rentang bobot sisi dalam graf yang akan digenerate
    num_nodes = 10
    min_weight = 0
    max_weight = 9

    # Generate kode hamilton
    hamilton = generate_hamilton_code(num_nodes, min_weight, max_weight)

    # Fungsi generate_hamilton_code mengembalikan tuple kode hamilton dan total bobot yang dilalui
    hamilton_code, total_weight = hamilton

    # Mencari nilai modulus dari total bobot
    modulus_value = 0
    while (modulus_value == 0):
        modulus_value = total_weight % 6
        total_weight = total_weight + 1

    # Mencari dan mencetak hasil akhir dari kombinasi kedua algoritma
    result = combine_digits(f"{totp:0{digits}}", f"{hamilton_code:0{digits}}", modulus_value)
    print("Kode OTP dari algoritma TOTP                          : " + str(totp))
    print("Kode OTP dari algoritma Graf Hamilton                 : " + str(hamilton_code))
    print("Nilai modulus dari total bobot yang dilalui siklus hamilton : " + str(modulus_value))
    print("Hasil OTP kombinasi TOTP dan Graf Hamilton            : " + str(result))
```

Program tersebut telah berjalan dengan baik. Beberapa penjelasan tercantum pada komentar program. Program juga akan dilampirkan pada referensi. Berikut adalah contoh hasil dari keberjalanan program:

```
[Running] python -u "c:\Users\HP\OTP-Matdis\modified-totp.py"
Kode OTP dari algoritma TOTP                          : 103339
Kode OTP dari algoritma Graf Hamilton                 : 897031
Nilai modulus dari total bobot yang dilalui siklus hamilton : 3
Hasil OTP kombinasi TOTP dan Graf Hamilton            : 103031

[Done] exited with code=0 in 0.398 seconds
```

```
[Running] python -u "c:\Users\HP\OTP-Matdis\modified-totp.py"
Kode OTP dari algoritma TOTP                          : 783325
Kode OTP dari algoritma Graf Hamilton                 : 728314
Nilai modulus dari total bobot yang dilalui siklus hamilton : 1
Hasil OTP kombinasi TOTP dan Graf Hamilton            : 728314

[Done] exited with code=0 in 0.429 seconds
```

```
[Running] python -u "c:\Users\HP\OTP-Matdis\modified-totp.py"
Kode OTP dari algoritma TOTP                          : 259547
Kode OTP dari algoritma Graf Hamilton                 : 620791
Nilai modulus dari total bobot yang dilalui siklus hamilton : 5
Hasil OTP kombinasi TOTP dan Graf Hamilton            : 259541

[Done] exited with code=0 in 0.403 seconds
```

IV. Pembahasan

Seperti yang sudah disampaikan sebelumnya, penambahan algoritma pembangkitan kode OTP menggunakan graf berupa sirkuit Hamilton ditujukan untuk menambah tingkat keamanan kode OTP. Meskipun kode OTP hanya berupa 6 digit, yang artinya sangat mudah untuk ditebak melalui *brute force*, tetapi kode OTP ini memiliki limitasi waktu. Sehingga untuk melakukan peretasan dibutuhkan pemahaman algoritma dibaliknya untuk dapat menebak kode OTP dengan benar.

Dengan ditambahnya algoritma graf untuk membangkitkan kombinasi kode OTP, hal yang harus dipecahkan peretas untuk dapat memahami algoritma akan semakin banyak dan rumit. Hal tersebut didasari oleh konsep banyaknya kemungkinan sirkuit Hamilton yang dapat dibentuk oleh graf lengkap. Dalam algoritma yang telah dibuat, terdapat algoritma pembuatan graf lengkap dengan simpul sebanyak sepuluh. Itu artinya banyaknya kombinasi kemungkinan sirkuit Hamilton sebagai berikut:

$$(n-1)!/2 = (10-1)!/2 = 181.440$$

Dengan asumsi untuk memecahkan 1 kemungkinan dibutuhkan waktu 0,1 detik maka untuk *brute force* dibutuhkan waktu 302,4 menit atau sekitar 5 jam. Tentu hal tersebut mengindikasikan waktu yang sangat lama. Apalagi jika dibandingkan dengan masa aktif kode OTP yang umumnya hanya berkisar 30 detik.

Padahal pada algoritma ini masih dikombinasikan dengan TOTP yang tentu saja juga akan menambah waktu peretasan. Belum juga mengenai kemungkinan total bobot sisi sirkuit Hamilton yang juga akan memengaruhi waktu peretasan.

Berdassarkan teori-teori yang ada algoritma tersebut seharusnya dapat meningkatkan tingkat keamanan kode OTP dari peretasan. Namun perlu diingat kembali bahwa tingkat keamanan suatu algoritma juga dipengaruhi beberapa hal atau faktor. Seperti penempatan algoritma tersebut dalam sebuah sistem, tingkat kerumitan kata kunci hash, kecanggihan dari sisi peretasan dan masih banyak hal. Algoritma ini masih sangat sederhana dan masih banyak kekurangan seperti dengan belum adanya testing secara langsung dari sisi peretas.

V. Kesimpulan

Algoritma pembangkitan OTP masih dapat terus dikembangkan untuk meningkatkan keamanan dari peretasan. Salah satunya adalah dengan menggunakan konsep graf dan teori bilangan. Banyaknya kemungkinan sirkuit Hamilton yang dapat dibentuk dari graf lengkap meningkatkan banyaknya hal yang harus dipecahkan peretas. Graf lengkap dengan 10 simpul saja mampu menambahkan kemungkinan sebanyak 181.440 kombinasi.

Tidak dipungkiri masih terdapat kekurangan dalam proses pembuatan makalah ini. Baik dari sisi pembuatan algoritma, pencarian referensi, maupun pengembangan ide dan solusi. Oleh karena itu, diharapkan terus adanya pengembangan mengenai peningkatan keamanan kode OTP ini. Seperti mengkombinasikan dengan teori-teori yang lain sehingga semakin meningkatkan banyaknya kombinasi yang terjadi. Mengingat kode OTP ini terus digunakan untuk hal-hal yang sangat krusial, penulis berharap dengan adanya makalah ini memeberikan sedikit manfaat bagi pembaca.

VI. Ucapan Terima Kasih

Dengan rendah hati, penulis memanjatkan puji syukur kepada Tuhan Yang Maha Esa atas ridho-Nya sehingga penulis dapat menyelesaikan makalah ini. Penulis mengucapakan kepada seluruh pigak yang telah turut membantu penulis dalam

menyusun dan menyelesaikan makalah ini. Kepada seluruh dosen pengampu mata kuliah IF2120 Matematika Dskrit Tahun 2023/2024 dan terutama Ibu Fariska Zakhralativa Ruskanda sebagai dosen pengampu kelas K2 yang telah mengajarkan penulis berbagai ilmu penting yang penulis butuhkan untuk menyusun makalah ini. Tidak luput juga terima kasih untuk para peneliti lain yang terlebih dahulu membahas mengenai materi terkait, sehingga penulis dapat melihat referensi terkait

REFERENSI

[1] G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.

[2] W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[3] H. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1985, ch. 4.

[4] B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[5] E. H. Miller, "A note on reflector arrays (Periodical style—Accepted for publication)," *IEEE Trans. Antennas Propagat.*, to be published.

[6] J. Wang, "Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication)," *IEEE J. Quantum Electron.*, submitted for publication.

[7] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

[8]

[9] dina_oktavia,+##default.groups.name.manager##,+247-99Z_Artikel-1052-1-2-20170614.pdf

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

Diana Tri Handayani
13522104