Analisis Penerapan Teori Bilangan dalam Sistem Verifikasi OTP (One Time Password)

Nathan Adhika Santosa - 13524041
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nathan.santosa37@gmail.com , 13524041@std.stei.itb.ac.id

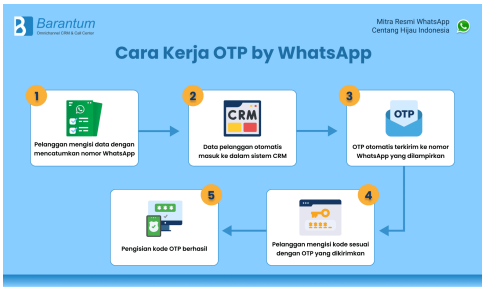
***Abstrak*— OTP (One Time Password) merupakan salah satu metode verifikasi yang sering digunakan karena bersifat unik, hanya sekali pakai, dan dikirim langsung dari penyedia layanan, sehingga sangat efektif untuk melawan kejahatan siber. OTP sangat berhubungan erat dengan matematika diskrit, khususnya teori bilangan. Konsep teori bilangan, seperti fungsi hash, modulo, dan kriptografi sering digunakan dalam pembuatan OTP. Konsep lain juga memainkan peran dalam OTP, seperti konsep operator logika dan kombinatorial pada algoritma SHA-1 selama proses HMAC, meskipun tidak menjadi fokus pembahasan.**

Keywords—Hash, Modulo, HOTP, TOTP

I. Pendahuluan

Seiring berjalannya waktu, adopsi digital semakin meningkat di berbagai bidang, mulai dari sektor keuangan, pemerintahan, pendidikan, bahkan pada tingkat individu. Aktivitas digital, seperti belanja online, perbankan digital, dan penggunaan cloud kini telah menjadi bagian dari kehidupan sehari-hari masyarakat di dunia. Namun, peningkatan adopsi digital juga menyebabkan meningkatnya jumlah kejahatan siber terutama pembobolan digital/*hacking* dimana terjadinya pencurian data atau penyalahgunaan identitas orang lain. Salah satu langkah untuk mengurangi jumlah pembobolan adalah dengan menggunakan OTP.

OTP (One Time Password) merupakan kode khusus sekali pakai yang mengandung deretan angka atau huruf. OTP biasanya dikirimkan melalui perantara-perantara, seperti sms, whatsapp, email, dll. OTP secara umum digunakan saat melakukan login pada suatu aplikasi, transaksi online, dan reset password. OTP sangat efektif dalam menghambat pembobolan karena kode unik yang hanya sekali pakai dan waktu penggunaan yang terbatas (misalnya 30 detik). Hal ini menyebabkan para *hacker* sulit untuk membobolkan sistem dengan *brute force* sehingga meskipun *hacker* memiliki email/username dan password Anda, *hacker* tetap tidak bisa mengakses akun Anda tanpa memiliki OTP.



Gambar 1.1 : Cara kerja OTP

(Sumber : https://www.barantum.com/blog/otp-by-whatsapp/)

Namun bagaimana cara kerja pembuatan kode OTP? Sistem verifikasi OTP bekerja karena dua komponen, yaitu OTP generator dan server autentikasi. OTP generator ini menggunakan konsep dasar matematika diskrit, terutama konsep teori bilangan, yaitu cabang matematika yang mempelajari sifat-sifat bilangan bulat. Oleh karena itu, pemahaman konsep teori bilangan sangat diperlukan untuk memahami cara kerja pembuatan OTP.

II. Landasan Teori

A. *Operator Bitwise/Operator Logika/Proposisi Majemuk*

Misalkan terdapat premis p dan q, maka

- Konjungsi (AND)
  p $\wedge$ q bernilai benar (1 dalam biner) jika p dan q bernilai benar
- Disjungsi (OR)
  p $\vee$ q bernilai benar (1 dalam biner) jika salah satu atau keduanya bernilai benar
- Ingkaran (~)
  ~p bernilai benar (1 dalam biner) jika bernilai salah (0 dalam biner)
- Disjungsi Ekslusif (XOR)
  p $\oplus$ q bernilai benar (1 dalam biner) jika dan hanya jika salah satunya bernilai benar, bukan dua-duanya.

B. *Teorema Euclidian*

Misalkan m dan n adalah dua bilangan bulat dengan syarat $n > 0$. Jika m dibagi dengan n maka terdapat dua buah

bilangan bulat unik q (*quotient*) dan r (*remainder*), sedemikian sehingga

$$m = nq + r$$

dengan $0 \leq r < n$.

Contoh :

100/6 = 16 sisa 4

dalam teorema Euclidian, bisa diubah menjadi bentuk:

100 = 6.16 + 4

C. *Aritmatika Modulo*

Misalkan a adalah bilangan bulat dan m adalah bilangan bulat > 0. Operasi a mod m (dibaca "a modulo m") memberikan sisa jika a dibagi dengan m. Dengan kata lain, a mod m = r sedemikian sehingga a = mq + r, dengan $0 \leq r < m$.

Contoh :

100 mod 6 = 4

D. *Fungsi Hash*

Data yang disimpan di memori perlu ditempatkan seefisien mungkin sehingga pencarian dapat dilakukan dengan cepat. Setiap data memiliki *"field"* unik yang membedakan suatu data dengan data yang lain. Fungsi hash ini digunakan untuk menempatkan data yang memiliki nilai kunci k. Fungsi hash yang paling umum digunakan dalam bentuk

$$h(k) = k \bmod m$$

Namun, fungsi hash, terutama dalam pembuatan kode unik, akan berbentuk berbeda dengan umumnya agar tidak mudah untuk ditebak atau dibobolkan.

Fungsi hash bukan fungsi dengan sifat satu-ke-satu, dimana nilai yang berbeda dapat menghasilkan nilai yang sama, maka hal ini dapat menyebabkan bentrokan dalam penempatan data. Maka dari itu, diperlukan kebijakan resolusi bentrokan, dimana terdapat banyak variasi namun resolusi yang paling umum digunakan adalah penempatan pada slot yang lebih tinggi.

Contoh :

Misal kita memiliki data 15, 21, 62, 34, dan 53 dan m = 5, maka

h(15) = 15 mod 5 = 0
h(21) = 21 mod 5 = 1
h(62) = 62 mod 5 = 2
h(34) = 34 mod 5 = 4
h(53) = 53 mod 5 = 3

E. *Kriptografi*

Kriptografi merupakan ilmu atau seni untuk menjaga keamanan pesan. Sebuah pesan diamankan dengan menyandikannya menjadi pesan yang tidak mempunyai makna. Kriptografi berguna untuk menyembunyikan informasi dari orang atau pihak yang tidak berhak untuk memperoleh informasi tersebut.

Pesan atau informasi yang ingin dikirimkan bernama plainteks (teks yang jelas dan dapat dimengerti), sedangkan pesan hasil disandikan disebut cipherteks. Cipherteks ini dapat dikembalikan menjadi plainteks/teks awalnya untuk orang yang berhak. Proses pengubahan plainteks menjadi cipherteks disebut dengan enkripsi, sedangkan kebalikannya disebut sebagai dekripsi.

Dalam notasi matematis, cipherteks dilambangkan dengan C dan plainteks dilambangkan dengan P, maka fungsi enkripsi E sebagai berikut

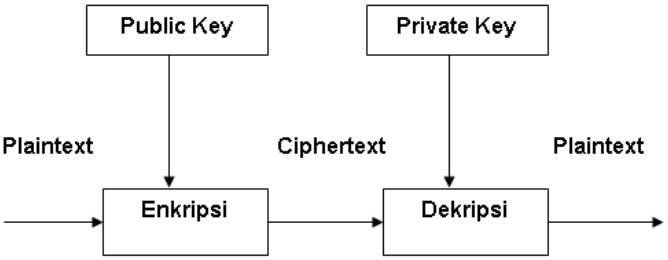
$$E(P) = C$$

Dan fungsi dekripsi D sebagai berikut

$$D(C) = P$$

Fungsi matematika yang disebut algoritma kriptografi digunakan untuk enkripsi dan dekripsi. Sebuah algoritma kriptografi disebut kuat jika waktu yang dibutuhkan untuk memecah data cipherteks menjadi plainteks lebih lama sehingga dapat dikatakan kuatnya algoritma kriptografi ekuivalen dengan waktu. Kekuatan kriptografi terletak pada kunci yang berupa deretan karakter atau bilangan bulat, kunci ini dapat dirahasiakan atau tidak sehingga orang-orang yang mengetahui kunci yang dapat melakukan enkripsi dan dekripsi. Algoritma kriptografi dengan kunci enkripsi dan dekripsi yang sama disebut juga algoritma simetri, sedangkan kunci enkripsi dan dekripsi yang berbeda disebut dengan algoritma nirsimetri.

Kriptografi memiliki dua aplikasi, yaitu pengiriman data melalui saluran komunikasi dan penyimpanan data di dalam storage. Untuk aplikasi OTP, kriptografi digunakan untuk membuat sebuah kunci untuk kode OTPnya



Gambar 2.1 : Cara kerja kriptografi
(Sumber : https://dosenit.com/jaringan-komputer/security-jaringan/kriptografi)

F. *Algoritma HOTP*

HOTP (HMAC-based OTP) adalah OTP yang berdasarkan pada HMAC, dimana HMAC kepanjangan dari *Hash-based Message Authentication Code*.

HMAC digunakan untuk memverifikasikan sebuah pesan yang dikirim melalui sebuah perantara, verifikasi apabila pesan yang dikirim telah diubah atau tidak dengan menggunakan fungsi hash (SHA-1 atau *Secure Hash Algorithm-1* merupakan algoritma yang paling sering digunakan untuk HMAC) dan MAC (Menggunakan kunci rahasia untuk melakukan verifikasi terhadap pesan). Fungsi hash akan berjalan terlebih dahulu terhadap pesan yang akan dikirim/terima lalu MAC akan menggunakan kunci rahasia dan algoritma MAC untuk melakukan verifikasi terhadap pesan tersebut. Jika hasil yang diperoleh dari HMAC tidak sama dengan pesan yang dikirim oleh server, maka autentikasi akan gagal.

SHA-1 merupakan algoritma hash yang rumit namun sering digunakan di dalam bidang kriptografi. Maka hanya dijelaskan secara singkat saja. Alur kerja SHA-1 dimulai dengan pertama-tama diberi *padding* atau ditambahkan bit-bit ekstra agar total panjangnya menjadi kelipatan 512 bit, dimana akan terdapat penambahan bit '1', diikuti sejumlah bit '0', dan diakhiri dengan 64 bit yang merepresentasikan panjang asli pesan. Lalu, lima register internal 32 bit diinisialisasi dengan nilai tetap. Pesan yang sudah di-*padding* dibagi menjadi blok-blok 512 bit, lalu blok-blok tersebut akan diproses secara berurutan sebanyak 80 putaran. Dalam setiap putaran, data blok diperluas dan dicampur dengan nilai register, konstanta putaran, serta berbagai operasi bitwise dan penambahan modulo. Setelah itu, nilai akhir dari register tersebut ditambahkan ke nilai hash total yang terakumulasi sehingga menghasilkan hash 160 bit unik.

HOTP didapatkan dengan memberi masukan, yaitu kunci rahasia dan *counter* (angka yang bertambah terus setiap kali sebuah OTP baru dibuat, hal ini dilakukan agar memastikan keunikan OTP setiap kali pembuatannya). Setelah menerima masukan tersebut, HMAC akan menghasilkan output sebesar 160 bit lalu diperpendek menjadi ukuran OTP yang diinginkan dengan menggunakan fungsi truncate HOTP. Algoritma HOTP didefinisikan sebagai berikut

$$HOTP(K,C)=Truncate(HMAC{-}SHA1(K,C))$$
atau
$$HMAC(K,m)=H((K'\oplus opad) \mathbin{//} H((K'\oplus ipad) \mathbin{//} m))$$

Keterangan :

K - Kunci rahasia bersama
C - Nilai counter
m - Pesan yang dikirim/counter
K' - K yang telah diproses menjadi 64-byte
ipad - byte 0x36 diulang
opad - byte 0x5c diulang
|| - Menggabungkan
H - Fungsi hash (biasanya SHA-1)

G. Algoritma TOTP

TOTP (*Time-based OTP*) dari namanya adalah *One Time Password* yang dibuat berdasarkan waktu. Secara garis besar, algoritma TOTP hampir sama dengan algoritma HOTP. Algoritma TOTP menggunakan fungsi hash (biasanya SHA-1), menerima masukan kunci rahasia bersama, dan parameter counter. Namun parameter *counter* pada HOTP diganti menjadi waktu (T) untuk algoritma TOTP. Persamaan algoritma TOTP sebagai berikut

$$TOTP = HOTP(K, T)$$

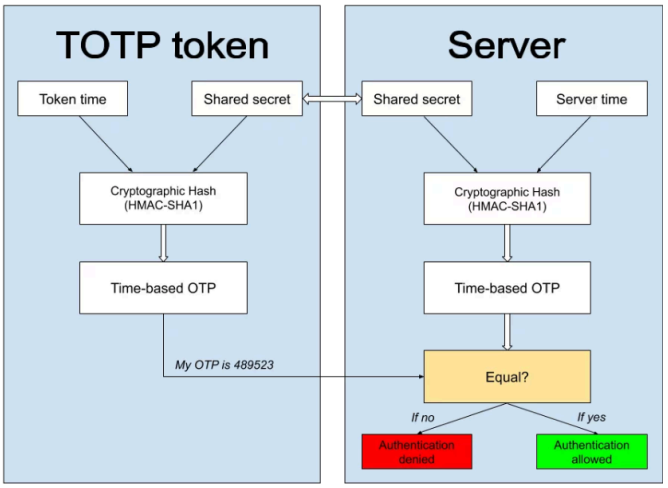
dengan T definisi sebagai berikut

$$T = (\text{Current Unix time} - T0) / X$$

Keterangan :

T0 : Waktu awal yang disepakati
X : Interval waktu yang telah disepakati

Algoritma TOTP jika dibandingkan dengan HOTP lebih "kuat" sebab sifatnya yang berbatas waktu, membuat kode OTP kedaluwarsa dengan cepat sehingga sangat efektif melawan serangan *replay*. Kemudahan dalam sinkronisasi berbasis waktu juga membuatnya lebih praktis untuk implementasi modern. Oleh karena itu, TOTP lebih sering digunakan daripada HOTP.



Gambar 2.2 : Proses Autentikasi TOTP
(Sumber : https://www.protectimus.com/blog/totp-algorithm-explained/)

III. PEMBAHASAN

Untuk melakukan analisis hubungan antara teori bilangan dengan OTP, sangat diperlukan contoh agar lebih mudah untuk dijelaskan. Namun, penulis tidak menggunakan contoh kode nyata, sebab perusahaan-perusahaan besar, seperti Google,

BCA, Tokopedia, dll tidak membagikan kode mereka ke publik. Sebab, jika mereka memberi kode mereka, maka sistem OTP mereka terdapat kemungkinan untuk dibobolkan lebih mudah oleh *hacker*. Oleh karena itu, penulis telah merancang kode python di visual studio code yang menggambarkan kode untuk pembuatan OTP secara umum, namun tanpa pengiriman dan verifikasi ke server hanya verifikasi ke lokal.

H. Analisis Implementasi OTP Menggunakan Bahasa Pemrograman Python

Berikut ini disajikan dua implementasi program Python untuk pembuatan OTP yang telah penulis rancang. Penulis membuat dua kode sebab OTP memiliki dua jenis yang telah dibahas, yaitu TOTP dan HOTP.

```python
import hmac
import hashlib
import base64
import struct
import random

def buat_otp(secret, counter, digits = 6):
    key = base64.b32decode(secret.upper())
    msg = struct.pack(">Q", counter)
    h = hmac.new(key, msg, hashlib.sha1).digest()
    offset = h[-1] & 0x0F
    potong = h[offset:offset+4]
    kode = struct.unpack(">I", potong)[0] & 0x7FFFFFFF
    return str(kode % 10**digits).zfill(digits)

secret = 'MYSECRETKEY23456'

counter = random.randint(0, 100000)

otp = buat_otp(secret, counter)
print("OTP yang dikirim ke pengguna:", otp)
masukan = input("Masukkan OTP: ")

if masukan == otp:
    print("OTP valid!")
else:
    print("OTP salah.")
```

Gambar 3.1 : Kode Python untuk HOTP

```
OTP yang dikirim ke pengguna: 183467
Masukkan OTP: 183467
OTP valid!
PS D:\ITBeh\coding> & C:/Users/Nathan
OTP yang dikirim ke pengguna: 136647
Masukkan OTP: 136647
OTP valid!
```

Gambar 3.2 : Hasil dari kode python HOTP

Di atas ini merupakan kode python untuk HOTP (HMAC-based OTP) dan hasilnya. Sebelum lanjut ke penjelasan kode dan hubungan antara HOTP dengan teori bilangan, terdapat beberapa catatan untuk program HOTP tersebut. Pertama, kunci rahasia yang penulis gunakan tidak random, kunci rahasia penulis adalah "MYSECRETKEY23456", sementara itu sebagian besar kunci yang digunakan oleh perusahaan umumnya bersifat acak. Kedua, counter yang seharusnya bertambah sebanyak satu setiap kali user membuat OTP baru tidak dilakukan disini, karena hal tersebut memerlukan penyimpanan data secara konsisten melalui file eksternal. Maka agar OTP tetap unik setiap kali dibuat, penulis menggunakan angka acak dari rentang 0 hingga 100000.

Pada kode tersebut terdapat sebuah fungsi yang bernama buat_otp dengan menerima parameter masukan secret (kunci rahasia), counter (untuk menjaga keunikan OTP), dan digits yang sudah didefinisikan (jumlah angka yang terdapat dalam OTP).

Di dalam fungsi buat_otp, didefinisikan sebuah variabel bernama "key" yang memastikan bahwa kunci rahasia "MYSECRETKEY23456" adalah string yang terdiri dari huruf kapital, angka (2-7), dan panjangnya dapat dibagi 8 sehingga sesuai dengan base32. Variabel "msg" mengubah counter dari bentuk integer menjadi format hexadecimal agar dapat digunakan oleh HMAC.

Variabel h menyimpan hasil dari HMAC yang menerima kunci rahasia yang telah diubah, msg, dan algoritma HMAC yang digunakan, yaitu SHA-1. Disinilah konsep teori bilangan fungsi hash bermain peran dengan algoritma SHA-1 dan begitu juga dengan konsep kriptografi dimana terjadinya enkripsi kunci rahasia dengan menggunakan fungsi hash. Dibawah ini merupakan algoritma HMAC,

$$HMAC(K,m)=H((K'\oplus opad) \mathbin{//} H((K'\oplus ipad) \mathbin{//} m))$$

algoritma HMAC yang menerima kunci rahasia dan counter melakukan dua kali hash yang dilambangkan dengan H, dimana di dalam fungsi hash terdapat perhitungan rumit yang terdiri dari operasi bitwise dan modulo sehingga memastikan input akan menghasilkan output yang unik(fungsi satu ke satu).

Pada variabel “offset”, diambilkan byte terakhir dari hasil HMAC yang berupa 20 byte dengan menggunakan operator bitwise AND. Selanjutnya, variabel “potong” menyimpan empat byte hasil ekstraksi dari HMAC yang dimulai dari posisi “offset”. Variabel “kode” mengambil empat byte dari “potong” dan mengubahnya menjadi unsigned integer (bilangan bulat non-negatif) dengan format big endian (Byte significant diberi pada alamat terendah).

Saat kode unik akan dikembalikan, nilai tersebut terlebih dahulu dikenai operasi modulo dengan 1000000. Modulo disini berperan dalam memastikan bahwa OTP terakhir tidak melebihi enam digit yang telah ditentukan terlebih dahulu. Misalnya, isi kode adalah 4294967295, maka modulo akan mengubah kode unik menjadi 967295. Apabila kode unik pada akhirnya tidak mengandung persis enam digit, maka “.zfill(digits)” akan mengisi depan kode unik dengan angka 0 hingga kode unik menjadi kode unik yang terdiri dari enam digit.

```python
import time
import hmac
import hashlib
import base64
import struct

def buat_totp(secret, interval=30, digits=6):
    key = base64.b32decode(secret.upper())
    counter = int(time.time()) // interval
    counter_bytes = struct.pack(">Q", counter)
    hmac_hash = hmac.new(key, counter_bytes, hashlib.sha1).digest()

    offset = hmac_hash[-1] & 0x0F
    truncated_hash = hmac_hash[offset:offset+4]
    kode = struct.unpack(">I", truncated_hash)[0] & 0x7FFFFFFF
    return str(kode % 10**digits).zfill(digits)

def verifikasi_totp(secret, user_input, interval=30, digits=6):
    current_otp = buat_totp(secret, interval, digits)
    return user_input == current_otp

secret = 'TEORIBILANGAN765'
generated_otp = buat_totp(secret)

print("Kode TOTP saat ini:", generated_otp)
user_input = input("Masukkan kode OTP: ")

if verifikasi_totp(secret, user_input):
    print("OTP benar!")
else:
    print("OTP salah.")
```

Gambar 3.3 : Kode Python untuk TOTP

```
PS D:\ITBeh\coding> & C:/Users
Kode TOTP saat ini: 774415
Masukkan kode OTP: 774415
OTP benar!
PS D:\ITBeh\coding> & C:/Users
Kode TOTP saat ini: 768100
Masukkan kode OTP: 768100
OTP benar!
```

Gambar 3.4 : Hasil dari Program TOTP

Diatas merupakan dua gambar, gambar pertama adalah program untuk membuat kode TOTP (*Time-base* OTP) dan gambar berikutnya adalah hasil dari program TOTP. Catatan untuk program tersebut adalah kuncinya yang tidak acak. Kunci rahasia di dalam program tersebut adalah “TEORIBILANGAN765”.

Proses pembuatan program TOTP secara garis besar sama dengan program HOTP, sebab

$$TOTP = HOTP(K, T)$$

dimana TOTP sama dengan HOTP namun parameter counter yang diterima oleh HOTP diganti dengan T. Hal ini berarti TOTP juga menggunakan fungsi hash yang bersifat satu ke satu pada proses HMAC(operator bitwise dan konstanta juga digunakan disini karena algoritma yang digunakan adalah SHA-1), kriptografi, dan modulo saat mengembalikan kode unik untuk memastikan kode unik berukuran kurang sama dengan enam digit.

Hal yang membedakan TOTP dan HOTP hanyalah parameter T, dimana T adalah

$$T = (\text{Current Unix time} - T0) / X$$

Maka terdapat perubahan kode pada variabel counter, dimana time.time() menghitung Current Unix time - T0 lalu dibagi dengan interval atau X yang didefinisikan sebagai 30 detik. Konsekuensi dari penggunaan parameter T adalah kode OTP yang tidak unik jika dihasilkan dalam rentang waktu tertentu.

```
PS D:\ITBeh\coding> & C:/Us
Kode TOTP saat ini: 205924
Masukkan kode OTP: 205924
OTP benar!
PS D:\ITBeh\coding> & C:/Us
Kode TOTP saat ini: 205924
Masukkan kode OTP: 205924
OTP benar!
```

Gambar 3.5 : Hasil Program TOTP dengan Jarak Waktu Pembuatan OTP yang Pendek

Diatas adalah hasil OTP jika penulis langsung membuat kode OTP baru. Pada gambar 3.4, penulis menunggu waktu terlebih dahulu sekitar 30-an detik untuk membuat kode OTP baru. Namun, pada implementasi kenyataan, hal ini tidak merupakan masalah sebab pembuatan OTP baru akan dibatasi waktu (Perlu menunggu X waktu untuk membuat OTP baru).

IV. Kesimpulan

Berdasarkan hasil analisis terhadap program pembuatan OTP secara umum, dapat disimpulkan bahwa algoritma HOTP dan TOTP yang merupakan dua jenis OTP memiliki keterkaitan erat dengan konsep teori bilangan, khususnya konsep fungsi hash, kriptografi, dan modulo.

Fungsi hash memiliki peran utama, terutama dalam menjaga keunikan OTP dengan penggunaan HMAC. Kompleksitas fungsi hash, seperti SHA-1, yang menggunakan operator bitwise dan perhitungan rumit membantu dalam menjaga keunikan dan sifat satu ke satu fungsi hash. Meskipun modulo tidak memiliki peran sebesar fungsi hash, modulo tetap memiliki peran yang cukup besar. Tanpa modulo, OTP tidak dapat dibatasi jumlah angka yang akan dikeluarkan.

Selain teori bilangan, OTP juga berhubungan dengan materi logika dan kombinatorial, meskipun tidak dibahas secara mendalam. Materi logika, khususnya operator bitwise, sangat sering digunakan untuk algoritma HMAC dan fungsi hash SHA-1. Lalu konsep kombinatorial hadir secara implisit, hadir dalam konteks jumlah kemungkinan OTP sehingga sebagai pertimbangan seberapa kuat terhadap serangan *brute-force*.

Dengan demikian, penggunaan konsep teori bilangan pada OTP, membuat OTP menjadi metode verifikasi yang kuat untuk digunakan terutama pada masa pesat perkembangan digital.

V. Lampiran

Berikut adalah source code dalam pembuatan program HOTP dan TOTP :
https://github.com/N8NAS/Analisis-Penerapan-Teori-Bilangan-dalam-Sistem-Verifikasi-OTP-One-Time-Password-.git

VI. Ucapan Terima Kasih

Terima kasih kepada Tuhan Yang Maha Esa karena telah memberkati penulis sehingga penulis dapat menyelesaikan makalah ini tanpa mengalami kendala yang terlalu berat. Penulis juga ingin mengucapkan terima kasih kepada Dr. Ir. Rinaldi Munir, M.T. selaku dosen pengampuh mata kuliah IF1220 kelas 1 karena telah membimbing dan mengajarkan kelas kami selama satu semester ini. Terakhir penulis juga ingin mengucapkan terima kasih kepada orang tua penulis karena telah memberi semangat kepada penulis selama proses pembuatan makalah ini.

Referensi

[1] R. Munir, "Teori bilangan bagian 1,", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/15-Teori-Bilangan-Bagian1-2024.pdf., diakses 18 Juni 2025 pukul 18.00.

[2] R. Munir, "Teori bilangan bagian 3,", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/17-Teori-Bilangan-Bagian3-2024.pdf., diakses 18 Juni 2025 pukul 18.00.

[3] R. Munir, "Logika,", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/01-Logika-2024.pdf., diakses 19 Juni 2025 pukul 20.00.

[4] "One Time Password Algorithm in Cryptography,", https://www.tutorialspoint.com/cryptography/one_time_password_algorithm_in_cryptography.htm., diakses 17 Juni 2025 pukul 19.00.

[5] "OTP adalah: Pengertian, Fungsi, dan Cara Kerja,", https://codingstudio.id/blog/otp-adalah/., diakses 17 Juni 2025 pukul 19.00.

[6] Kathleen Richards, Ivy Wigmore, "One-time password (OTP),", https://www.techtarget.com/searchsecurity/definition/one-time-password-OTP, diakses 17 Juni 2025 pulul 19.00.

[7] Maxim Oliynyk, "HOTP Algorithm Explained,", https://www.protectimus.com/blog/hotp-algorithm/. diakses 17 Juni 2025 pukul 19.00.

[8] "One Time Password (OTP) Algorithm in Cryptography,", https://www.geeksforgeeks.org/computer-networks/one-time-password-otp-algorithm-in-cryptography/., diakses 17 Juni 2025 pukul 20.00.

[9] B. N. Tan, "Understanding OATH HOTP Authentications," https://bntan.medium.com/understanding-oath-hotp-authentications-42dc6012c41d., diakses 18 Juni 2025 pukul 16.00.

[10] "SHA-1 Hash in Java,", https://www.geeksforgeeks.org/sha-1-hash-in-java/., diakses 18 Juni 2025 pukul 17.00.

[11] Maxim Oliynyk, "TOTP Algorithm Explained,", https://www.protectimus.com/blog/totp-algorithm-explained/., diakses 18 Juni 2025 pukul 18.00.

[12] Aryaman Sharda, "How Do Time-based One-Time Password (TOTP) Services Work?,", https://digitalbunker.dev/how-do-time-based-one-time-password-totp-services-work/., diakses 18 Juni 2025 pukul 19.00.

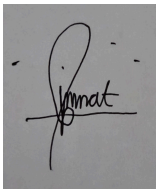
[13] "SHA-1,", https://en.wikipedia.org/wiki/SHA-1. diakses 19 Juni 2025 pukul 14.00.

[14] M'Raihi, et al., "RFC 6238 - TOTP: Time-Based One-Time Password Algorithm,", https://datatracker.ietf.org/doc/html/rfc6238. diakses 18 Juni 2025 pukul 22.00

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, 20 Juni 2025



Nathan Adhika Santosa
13524041