Analisis Perbedaan HOTP dan TOTP dalam Implementasi dan Keamanannya

Julius Arthur – 13523030[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*juliusarthur2005@gmail.com*, *13523030@std.stei.itb.ac.id*

***Abstract*— This paper analyzes the differences between HMAC-based One-Time Password (HOTP) and Time-based One-Time Password (TOTP), two key algorithms used in Two-Factor Authentication (2FA) systems. While both are derived from HMAC (Hash-based Message Authentication Code) and ensure secure generation of one-time passwords, they differ significantly in their implementation. HOTP generates codes based on a counter, but requires careful counter synchronization to avoid failures. TOTP, in contrast, uses time as a dynamic variable, producing codes that automatically expire after a predefined interva. This paper explores their fundamental principles, and synchronization requirements.**

***Keywords*—HOTP, TOTP, HMAC, 2FA**

I. PENDAHULUAN

Dalam era digital, keamanan data menjadi isu penting yang wajib diperhatikan baik bagi pengguna maupun penyedia layanan digital. Ancaman terhadap keamanan data pun meningkat seiring berkembangnya teknologi. Kata sandi menjadi hal yang lazim digunakan untuk melindungi data dari serangan yang mungkin terjadi. Namun, serangan terhadap data yang dilindungi oleh sebuah kata sandi masih sering terjadi dan berhasil. Maka, dibutuhkan sebuah atau beberapa proses autentikasi lanjutan.

Autentikasi multifaktor (MFA; two-factor authentication/ 2FA) merupakan metode untuk memberi perlindungan data yang lebih tinggi. Dengan menggunakan MFA, pengguna diwajibkan untuk memberikan kombinasi dari sesuatu yang dimiliki pengguna, sesuatu yang diketahui pengguna (password/ PIN), serta identifikasi pengguna (biometrik). Pengguna wajib memberikan 2 atau lebih identifikasi (faktor) yang berbeda jenis untuk mengakses suatu data[1]. Setelah memberikan password, pengguna perlu memberikan identifikasi tambahan yang akan lebih menjamin keamanan.

Autentikasi multifaktor telah digunakan dalam berbagai bidang, terutama berhubungan dengan data sensitif. Faktor yang digunakan dalam mengindentifikasi pengguna sudah diterapkan dalam berbagai bentuk. Beberapa contoh faktor adalah Flasdisk, kartu ATM, password, PIN, serta biometrik seperti sidik jari, iris mata, serta wajah. Sebagai contoh, dalam ATM, pengguna perlu menggunakan kartu yang tepat serta PIN yang tepat untuk mengakses rekening. Dalam hal ini, kartu dan PIN menjadi faktor yang dikombinasikan dalam menjaga keamanan rekening bank.

Salah satu penerapa 2FA yang paling sering digunakan adalah autentikasi berbasis ponsel. Untuk mengakses sebuah data, pengguna perlu memberikan password serta sebuah kode valid yang berubah – ubah (OTP). Kode ini biasanya terdiri dari 4 – 6 digit yang nilainya dinamis setiap kali pengguna berusaha untuk mengakses data tersebut. Kode ini dapat dikirim kepada pengguna melalui SMS atau aplikasi pihak ketiga.

MFA menggunakan algoritma matematika yang kompleks untuk menghasilkan kode yang aman. Dalam hal ini, teori bilangan serta kombinatorika bekerja untuk memastikan kode untuk autentikasi adalah acak dan mengurangi kemungkinan berhasilnya serangan menggunakan *brute force*. Makalah ini akan membahas penerapan teori bilangna dan kombinatorika dalam 2FA/MFA, serta fokus pada algoritma Time-based One Time Password (TOTP) dan HMAC-based One Time Password dalam menghasilkan kode pada 2FA berbasis ponsel.

II. ALGORITMA HMAC

HOTP (HMAC-based One Time Password) adalah model OTP berbasis HMAC (hash- based message authentication code). HOTP dipublikasikan pada IETF RFC 4226 pada 2005.

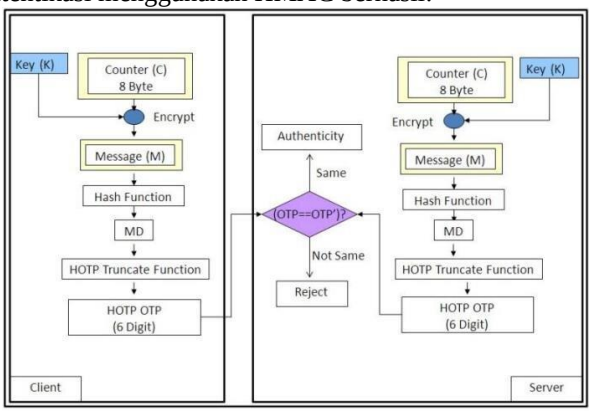
Fungsi hash menjadi salah satu fundamental pada algoritma HOTP. Fungsi hash merupakan sebuah fungsi yang memproses sebuah data dengan panjang bebas menjadi sebuah nilai dengan panjang yang tetap. Fungsi hash merupakan fungsi satu arah, mengubah input menjadi hasil dan tidak bisa sebaliknya. Salah satu fungsi hash yang populer pada kriptografi adalah SHA (Secure Hash Algorithm). Sebagai contoh, SHA-1, juga digunakan dalam algoritma HMAC, mengubah sebuah data menjadi hasil dengan besar 160 bit.

HMAC merupakan algoritma yang menggunakan fungsi hash SHA-1, SHA-2, dan lain- lain, yang dikombinasikan dengan sebuah kunci rahasia. HMAC menyediakan keamanan enkripsi layaknya *digital signature* dan *message authetication code* (MAC) lainnya. Keduanya memastikan integritas data serta autentikasi dari pesan yang akan diproses. Namun, HMAC menggunakan *symetric keys*, sedangkan *digital signature* menggunakan *assymetric keys*.

Secara umum, HMAC mendapatkan hasil dengan menerapkan fungsi hash sebanyak dua kali. Kunci yang digunakan pada HMAC diproses menjadi dua buah kunci (*outer key* dan *inner key*). Fungsi hash diterapkan pada kombinasi *inner*

key dan pesan yang ingin diproses. Kemudian, fungsi hash diterapkan lagi pada hasil hash pertama dengan *outer key*. Proses ini dapat mencegah HMAC dari resiko serangan kriptografi *lenght extension attack*. Serangan ini memanfaatkan hasil hash pesan1 dan panjang dari pesan1 untuk mengetahui hasil hash pada pesan1 + pesan2.

HMAC bekerja pada kedua sisi dari perangkat lunak. Baik client dan server melakukan fungsi hash pada pesan dan kunci yang sama. Client akan mengirim hasil hash dan server akan melakukan proses hash yang sama. Jika keduanya identik, maka autentikasi menggunakan HMAC berhasil.



Gambar 1 perhatikan bahwa baik client dan server harus memiliki pesan dan kunci yang sama untuk autentikasi yang sukses.

Didefinisikan dua string tetap, yaitu *inner padding* (ipad) dan *outer padding* (opad).

$$ipad = the\ byte\ 0x36\ repeated\ B\ times.$$
$$opad = the\ byte\ 0x5C\ repeated\ B\ times.$$

HMAC didefinisikan dengan:

HMAC(K, m) = H(Kj XOR opad || H(Kj XOR ipad || m)

dengan H sebagai fungsi hash, || adalah konkatenasi, K adalah kunci, dan m adalah pesan yang ingin diautentikasi. Kj diperoleh dengan memperpanjang K hingga ukuran nya sama dengan ipad atau opad. Jika panjang K adalah 20 byte dan B adalah 64, maka K akan dikonkatenasi dengan 24 byte 0 hingga panjang K adalah 64 byte.[2]

IV. ALGORITMA HOTP

HOTP adalah salah satu jenis password sekali pakai (OTP) berbasis algoritma HMAC. Algoritma ini menghasilkan sebuah password yang mudah digunakan oleh manusia dalam satu kesempatan saja. HOTP menggunakan sebuah variabel yang dinamis (*counter*) dan kunci simetris untuk menghasilkan kode sekali pakai.

Variabel penting yang diperlukan untuk autentikasi menggunakan HOTP adalah

H: fungsi hash
K: kunci rahasia
C: *counter*
d: panjang hasil HOTP (jumlah digit kode autentikasi yang diinginkan).

Algoritma HOTP didefinisikan sebagai

HOTP(K,C) = Truncate(HMAC(K,C))

atau

HOTP(K,C) = Truncate(HMAC-SHA-1(K,C))

jika menggunakan SHA-1 sebagai fungsi hash pada algoritma HMAC, dengan

Truncate: fungsi yang mengubah hasil HMAC-SHA-1 menjadi nilai dengan panjang yang sesuai dengan HOTP yang telah disepakati. Truncate dilakukan dengan sebuah fungsi DT (*dynamic trucate*)

Algortima HOTP dimulai dengan menghasilkan sebuah nilai dari HMAC-SHA-1(K,C), hs, dengan panjang 160 bit atau 20 byte string. Kemudian, diterapkan fungsi DT pada hs,

S = DT(hs).

Hasil *dynamic truncate*, S, dikonversi menjadi integer, Snum. DT dimulai dengan mengambil 4 bit terakhir dari hs sebagai offset. Kemudian ambil 8 byte pada hs mulai dari indeks offset, P = hs[offset] ... hs[offset + 3]. Sebanyak 31 bit terakhir adalah hasil dari *dynamic truncate*.

Hasil modulo Snum dengan 10^d adalah kode hasil HOTP.

Sebagai contoh, hasil HMAC-SHA-1 adalah

|00|01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|
|1f |86|98|69|0e|02|ca|16|61|85|50|ef|7f|19 |da|8e|94|5b|55|5a|

dengan baris pertama sebagai indeks byte dan baris kedua adalah nilai byte dari hasil hash.

Offset adalah 4 bit terakhir dari byte terakhir nilai hash (0x5a), yaitu 0xa (10 dalam desimal). Maka, ambil 4 byte dari indeks ke 10, 0x50ef7f19. Ambil hanya 31 bit terakhir untuk mencegah terjadinya konflik antara signed dan unsigned integer, sebagai P. Hasil modulo P dengan 10^6 akan menjadi hasil HOTP dengan panjang 6 digit (000000 – 999999).[3]

Pada setiap percobaan HOTP, client meningkatkan nilai counter C sebanyak 1, dan menghitung nilai HOTP berdasarkan nilai C yang baru. Jika hasil HOTP antara client dan server sama, pengguna berhasil melakukan autentikasi dan server meningkatkan nilai C pada server sebesar 1.

Masalah akan muncul jika terjadi percobaan gagal pada proses autentikasi, yaitu nilai C pada client berbeda dengan C pada server. Ini dapat disebabkan oleh percobaan HOTP yang salah oleh client, sehingga nilai c pada *client* bertambah 1. Dalam kasus ini, server akan membuka sebuah look-ahead window sebesar s dan menjalankan *resynch protocol*. Pada proses ini, server akan memeriksa nilai HOTP sebanyak s mulai dari conter C hingga C + s. Jika tidak ditemukan nilai HOTP yang sama dengan client, proses autentikasi dinyatakan gagal dan server akan menerima nilai HOTP yang baru dari client. Kerentanan keamanan terdapat pada proses *truncate* pada HMAC-SHA-1 yang dapat menurunkan keamanan fungsi

hash. Maka perlu diatur maksimal percobaan autentikasi pada server. Percobaan yang melebihi batas akan membuat pengguna terkunci dari mengakses data. [3] Terdapat alternatif lain dalam menangani percobaan HOTP yang salah.

IV. ALGORITMA TOTP

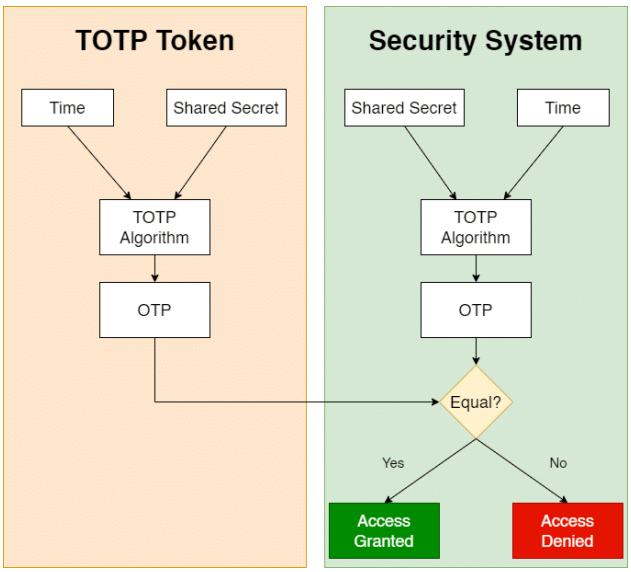
TOTP (Time-based One Time Password) merupakan sebuah metode password sekali pakai berbasis waktu. Algoritma ini merupakan salah satu turunan dari algoritma HOTP yang mengutamakan waktu sesaat sebagai salah satu parameternya. Secara umum persamaan algoritma TOTP mirip dengan HOTP, hanya saja TOTP menggunakan waktu sebagai pengganti counter. TOTP dapat menggunakan HMAC-SHA-256 dan HMAC-SHA-512 sebagai fungsi hash, menggantikan HMAC-SHA-1 yang menjadi standar pada algoritma HOTP [2]. Persamaan algoritma TOTP adalah

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

dengan T sebagai selisih waktu pengujian dengan T_0,

$$T = \frac{\text{Current Unix time} - T0}{x}$$

T biasanya didapatkan dengan melakukan pembulatan kebawah. Disini, X adalah *time step*(detik). Jika X adalah 30 detik dan T_0 adalah 0, maka selama detik ke 31 – 59, nilai T adalah sama, yaitu 1. [4]



Gambar 2 Proses autentikasi menggunakan TOTP

Waktu yang tercatat oleh client untuk menghasilkan sebuah OTP dapat saja berbeda saat server menerima permintaan validasi dari client. Karena alasan jaringan, perbedaan waktu ini dapat menjadi sangat besar, hingga server dan client berada pada window yang berbeda (nilai T yang berbeda). Tentu dengan nilai T yang berbeda, kode OTP yang dihasilkan oleh client akan berbeda dengan kode OTP yang dihasilkan pada server.Oleh karena itu, server perlu mencatat satu atau beberapa kode OTP yang dihasilkan pada window sebelumnya.

Waktu yang tercatat pada server dan client di saat yang sama dapat saja berbeda. Jika dibiarkan, client dan server dapat terus berada pada window yang berbeda, sehingga client tidak akan pernah berhasil melaukan autentikasi. Maka, perlu diberlakukan sebuah limit, berapa *time step* perbedaan antara client dan server yang diperbolehkan. Sebagai contoh, jika X = 30 detik, maka window yang diperbolehkan adalah window pada rentang T_0 hingga T_0 – 60 detik.[4]

IV. PENERAPAN HOTP DAN TOTP PADA PYTHON

Berikut adalah penerapan HOTP dan TOTP menggunakan bahasa python. Terdapat 4 fungsi utama yang digunakan yaitu, hmac, truncate, hotp, serta totp. Secara umum, cara kerja kode dibawah sama dengan algoritma yang sudah dijelaskan pada bagian sebelumnya.

HMAC membutuhkan sebuah kunci dan counter baik pada HOTP dan TOTP. Pada HOTP, counter dimulai dari angka 0 dan bertambah 1 seiring permintaan kode OTP yang baru. Sedangkan pada TOTP, counter bergantung pada selisih waktu pengujian dengan T_0 dan durasi. Pada kode ini, selisih waktu pengujian dengan T_0 diterapkan menggunakan fungsi time() pada library time pada python. Fungsi ini secara otomatis menghitung selisih waktu dengan *Epoch*.

Kode ini menggunakan SHA-1 sebagai fungsi hash pada HMAC. Byte dari kunci *key* perlu dilakukan konkatenasi dengan byte 0 agar mencapai panjang yang sesuai. Ini diimplemtasikan menggunakan fungsi ljust. Fungsi ljust akan mengisi kekosongan pada key dengan byte 0 hingga panjang/ ukuran dari key menjadi dengan ukuran blok yang ditentukan.

```python
def hmac(key, counter):
    ipad = bytes((x ^ 0x36) for x in range(256))
    opad = bytes((x ^ 0x5c) for x in range(256))

    kpad = key.ljust(block_size, b'\0')
    i_kpad = kpad.translate(ipad)
    o_kpad = kpad.translate(opad)
    hash_1 = hashlib.new("sha1", i_kpad + counter).digest()
    hash_2 = hashlib.new("sha1", o_kpad + hash_1).digest()

    return hash_2
```

Gambar 3 Fungsi HMAC menggunakan fungsi hash SHA-1

```python
def truncate(h):
    offset = h[-1] % 16
    truncated = int.from_bytes(h[offset:offset + 4], byteorder = 'big', signed = False) % 2 ** 31
    return truncated
```

Gambar 4 Fungsi truncate yang memotong hasil HMAC dengan offset yang sesuai

```python
def hotp(counter):
    h = hmac(base64.b32decode(key), counter.to_bytes(8, byteorder = 'big'))
    truncated = truncate(h)
    hotp_res = truncated % 10 ** digit
    return hotp_res
```

Gambar 5 Fungsi HOTP yang menghasilkan kode OTP sepanjang "digit" digit. Seluruh byte dibaca dan diproses menggunakan big Endian.

```python
def totp():
    h = hmac(base64.b32decode(key), int(time.time()/duration).to_bytes(8, byteorder = 'big'))
    truncated = truncate(h)
    totp_res = truncated % 10 ** digit
    return totp_res
```

Gambar 6 Fungsi TOTP yang menghasilkan kode OTP sepanjang "digit"digit. Seluruh byte dibaca dan diproses menggunakan big Endian. Selisih waktu didapat menggunakan time.time().

Hasil dari implementasi diatas dicocokkan dengan aplikasi autentikasi dari Google, *Authenticator*. Kunci yang digunakan pada kedua sistem adalah “SUPERSECRETKEYZZ” (tanpa tanda petik). Pada program, juga sebagai setelan otomatis pada *Authenticator*, durasi window pada TOTP adalah 30 detik. Dilakukan percobaan sebanyak 5 kali berturut- turut pada program dan *Authenticator*. Hasil dari kode implemntasi pada python adalah

```python
for i in range(len(hotp_values)):
    hotp_res = hotp(i)
    print('C =  ' + str(i + 1) + ', HOTP = '+ str(hotp_res))
```

```
C =  1, HOTP = 196901
C =  2, HOTP = 609131
C =  3, HOTP = 16996
C =  4, HOTP = 886247
C =  5, HOTP = 930790
```

Gambar 7 Hasil HOTP pada program

Dengan aplikasi *Authenticator*, menggunakan key yang sama, didapat hasil (dengan counter dari 0 hingga 4):

Counter	HOTP
0	196 901
1	609 131
2	016 996
3	886247
4	930 790

Tabel 1 Hasil HOTP pada Authenticator

Dan hasil dari algoritma TOTP adalah

```python
t_val = totp()
print("TOTP: ", str(t_val))
time.sleep(30)
t_val = totp()
print("TOTP: ", str(t_val))
time.sleep(30)
t_val = totp()
print("TOTP: ", str(t_val))
time.sleep(30)
t_val = totp()
print("TOTP: ", str(t_val))
time.sleep(30)
t_val = totp()
print("TOTP: ", str(t_val))
```

```
TOTP:  270118
TOTP:  16769
TOTP:  273541
TOTP:  967552
TOTP:  636500
```

Gambar 8 Hasil TOTP pada program

Dengan aplikasi *Authenticator*, menggunakan key yang sama, didapat hasil (dengan counter dari 0 hingga 4):

Counter	HOTP
0	196 901
1	016 769
2	273 451
3	967 552
4	636 500

Tabel 2 Hasil TOTP pada Authenticator

Terlihat, hasil algoritma HOTP dan TOTP pada program identik dengan hasil OTP dari aplikasi *Authenticator*

.

V. HOTP DAN TOTP

HOTP dan TOTP adalah algoritma yang memiliki tujuan sama, menghasilkan sebuah kode sekali pakai (OTP). Keduanya sama- sama didasari algoritma hash pada pesan yang sama, yaitu HMAC. Keduanya menjadi metode yang digunakan dalam autentikasi multi faktor. Namun, terdapat perbedaan antara HOTP dan TOTP.

Salah satu tujuan utama dari algoritma HOTP adalah meningkatkan keamanan pada alat penyimpanan data yang besar dan sensitif, seperti USB.

Terdapat beberapa poin utama dalam pengembangan algoritma HOTP. HOTP dapat bekerja pada perangkat yang tidak menerima input numerik. Algoritma ini juga menghasilkan sebuah kode yang mudah dibaca dan digunakan. Maka, panjang hasil HOTP harus mengedepankan kemudahan pemakaian[3].

HOTP menggunakan counter yang meningkat secara teratur. Namun, dapat terjadi ketidaksinkronan antara counter pada client dan counter pada server. Proses sinkronisasi, sudah dijelaskan dibagian sebelumnya menjadi sebuah bagian yang penting, sebab counter pada client terus bertambah setiap pemanggilan algoritma HOTP. Sedangkan pada server, counter bertambah jika dan hanya jika terjadi autentikasi yang berhasil.

Sedangkan pada TOTP, terdapat beberapa poin penting dalam pengembangannya.

Pada TOTP, baik client dan server harus dapat mengetahui waktu Unix pada tiap autentikasi. Waktu unix adalah rentang waktu yang telah berlalu dari 1 Januari 1970, dalam detik.

Selain itu, pada TOTP, durasi window, X, akan menentukan kevalidan dari sebuah OTP. Maka, client dan server harus memiliki beberapa parameter yang sama, salah satunya adalah X.

Perbedaan utama antara HOTP dan TOTP adalah cara counter bekerja dan waktu hidup sebuah OTP. HOTP adalah algoritma berbasis kejadian. OTP yang valid *hanya* berubah jika sudah terjadi autentikasi yang berhasil. Dalam arti lain, sebuah OTP dapat valid tanpa ada batasan waktu. Kode tersebut valid hingga digunakan. Hal ini sangat berbeda dengan nilai counter pada TOTP. Sebuah kode OTP hanya valid pada suatu rentang tertentu (misal 30 detik). Perbedaan ini menghasilkan faktor keamanan yang sangat berbeda. Pada HOTP, jika penyerang, oleh suatu cara, mengetahui kode OTP dari suatu perangkat, penyerang dapat menggunakannya kapanpun selama kode

tersebut tidak dipakai. Dalam arti lain, penyerang dapat mengakses suatu data sensitif dengan leluasa. Pada TOTP, pun penyerang mengetahui kode OTP, informasi tersebut hanya akan berguna untuk suatu waktu yang singkat saja (umumnya 30 detik). Penyerang tetap dapat mengakses data yang dilindungi, tetapi dalam waktu singkat saja.

V. KESIMPULAN

Teori bilangan merupakan sebuah cabang matematika yang telah berkembang sangat lama. Teori ini sudah diimplementasikan dalam berbagai bidang di kehidupan. Salah satu penerapannya adalah dalam bidang komputasi. Pembangkit bilangan acak menjadi hal yang penting dalam dunia digital. Bidang ini diperlukan terutama dalam melindungi data. Namun, seiring perkembangan teknologi, cara untuk menyerang data yang dilindungi pun juga berkembang. Pembangkit bilangan acak sederhana tidaklah cukup dalam melindungi hingga mengenkripsi data.

Fungsi HMAC, sebuah fungsi hash menjadi bagian penting dalam banyak algoritma perlindungan data. Dua jenis algoritma tersebut adalah HOTP dan TOTP. Keduanya dimanfaatkan dalam metode perlindungan data multi faktor autentikasi. Pengguna memerlukan lebih dari password untuk mengakses sebuah data, OTP. Kedua algoritma ini dapat menghasilkan kode OTP dengan dasar bekerja yang sama. Keduanya menggunakan algoritma HMAC dalam menghasilkan bilangan/ string acak. Perbedaan utamanya adalah sebuah counter yang berbasis kejadian dan berbasis waktu. TOTP cenderung lebih aman dibandingkan HOTP. Kode OTP pad TOTP hanya valid untuk suatu waktu yang singkat. Server juga harus melakukan validasi antara waktu pada server dan waktu Unix pada client. Namun, hal ini membawa sebuah tantangan bagi penggunannya. Pada TOTP, pengguna harus bisa membaca dan memasukkan kode, umumnya 6 digit, dalam waktu yang cukup singkat. Jika terdapat kendala, seperti kecepatan internet, serta delay yang cukup besar, kode OTP tersebut bisa menjadi tidak valid. Di sisi lain, kode OTP dari HOTP akan selalu valid hingga kode tersebut digunakan. Hal ini membawa dampak baik dan buruk. Pengguna dapat mengingat kode tersebut dan menggunakan kapanpun ia mau, tanpa ada tekanan waktu, delay, serta pengaruh kecepatan internet. Namun, disisi lain, jika kode OTP ini diketahui pihak lain, kode tersebut dapat dimanfaatkan oleh penyerang dengan lebih leluasa.

Kedua algoritma OTP tetap menghasilkan sebuah keamanan lebih. Menggunakan kode OTP akan selalu lebih aman dibandingkan tanpa OTP. Maka, autentikasi multi faktor menjadi hal yang krusial di era digital. Telah terdapat berbagai aplikasi yang mendukung penggunaan multi faktor autentikasi berbasis ponsel. Dengan ini, pengguna dapat semakin lebih mudah dalam meningkatkan kemanan data. Tingkat penyerangan terhadap data akan semakin sulit dan dapat menurun.[5]

REFERENCES

[1] David Temoshok, “Digital Identity Guidelines,”. NIST, 2024, pp : McGraw-Hill, 1964, pp. 25–26.
[2] H. Krawczyk, *HMAC: Keyed-Hashing for Message Authentication*. 1997.
[3] D. M'Raihi, *HOTP: An HMAC-Based One-Time Password Algorithm*. 2005.
[4] D. M'Raihi, *TOTP: Time-Based One-Time Password Algorithm*. 2011.
[5] NIST, https://www.nist.gov/itl/smallbusinesscyber/guidance-topic/multi-factor-authentication, diakses pada 5 Januari 2024

GAMBAR

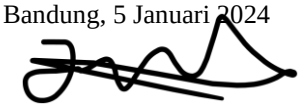
[1]https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FHow-HOTP-works-Lina-et-al-2021_fig8_377965404&psig=AOvVaw2SyRsMx2zXZlh06y4EnlRq&ust=1736345870507000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCPC9mMjm44oDFQAAAAAdAAAAABAE

[2]https://www.google.com/search?q=totp+algorithm&sca_esv=3431e2d2e020c503&rlz=1C1GCEB_enID909ID909&udm=2&biw=1920&bih=911&sxsrf=ADLYWIIvKAhgy59BGtDU9MePyeCR8-D3hw%3A1736343610641&ei=OoB-Z-jsJoeeseMPzJD2uQY&oq=totp+algor&gs_lp=EgNpbWciCnRvdHAgYWxnb3IqAggAMgUQABiABDIEEAAYHjIEEAAYHjIEEAAYHjIEEAAYHjIEEAAYHjIEEAAYHjIEEAAYHjIEEAAYHjIEEAAYHkibJ1CcAVjjIXACeACQAQCYAYEBoAGHBaoBAzIuNLgBA8gBAPgBAZgCCKACpgXCAgoQABiABBhDGIoFwgINEAAYgAQYsQMYgwEYCsICBhAAGAcYHsICBxAAGIAEGArCAgYQABgDGAqYAwCIBgGSBwMyLjagB_4e&sclient=img#vhid=rycf0AEh2eZpMM&vssid=mosaic

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, 5 Januari 2024



Julius Arthur 13523030