Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 12, No. 4, Agustus 2025, hlm. 799-808 e-ISSN: 2528-6579

IMPLEMENTASI *TIME-BASED ONE-TIME PASSWORD* MENGGUNAKAN ALGORITMA PHOTON UNTUK AUTENTIKASI DUA FAKTOR

Amry Yahya[1], Ari Kusyanti*[2], Primantara Hari Trisnawan[3]

[1,2,3]Universitas Brawijaya, Malang
Email: [1]amryyahya07@gmail.com, [2]ari.kusyanti@ub.ac.id, [3]prima@ub.ac.id
*Penulis Korespondensi

(Naskah masuk: 29 November 2024, diterima untuk diterbitkan: 27 Agustus 2025)

Abstrak

Di era digital yang makin maju, perlindungan terhadap data sensitif menjadi sangat penting. *Two-factor Authentication* (2FA) atau autentikasi dua faktor adalah metode keamanan yang efektif untuk memastikan bahwa hanya pengguna sah yang dapat mengakses data atau sistem sensitif dengan mengharuskan pengguna untuk memberikan dua bentuk identifikasi yang berbeda. Salah satu metode 2FA yang banyak digunakan adalah *Time-based One-Time Password* (TOTP) yang menggunakan algoritma *Hash-based Message Authentication* (HMAC) dengan fungsi *hash* SHA-1. Namun, fungsi SHA-1 diketahui memiliki kelemahan keamanan. Penelitian ini bertujuan untuk meningkatkan keamanan TOTP dengan mengimplementasikan fungsi *hash* PHOTON, algoritma *hash* ringan yang dirancang dengan keamanan yang baik dan penggunaan sumber daya komputasi yang efisien. Metodologi penelitian ini melibatkan pengembangan dan pengujian sistem autentikasi dua faktor berbasis TOTP dengan algoritma HMAC yang menggunakan fungsi *hash* PHOTON. Hasil penelitian menunjukkan bahwa sistem mampu bertahan dari *brute-force attack* dan *birthday attack*. Selain itu, fungsi TOTP yang menerapkan PHOTON memiliki waktu eksekusi yang lebih cepat dari SHA-1 dan SHA-2.

Kata kunci: *Autentikasi dua faktor, TOTP, PHOTON hash, HMAC*

IMPLEMENTATION OF TIME-BASED ONE-TIME PASSWORD USING PHOTON ALGORITHM FOR TWO-FACTOR AUTHENTICATION

Abstract

In the increasingly advanced digital era, protection of sensitive data is very important. Two-factor Authentication (2FA) is an effective security method to ensure that only authorized users can access sensitive data or systems by requiring users to provide two different forms of identification. One of the widely used 2FA methods is Time-based One-Time Password (TOTP) which uses the Hash-based Message Authentication (HMAC) algorithm with the SHA-1 hash function. However, the SHA-1 function is known to have security weaknesses. This study aims to improve the security of TOTP by implementing the PHOTON hash function, a lightweight hash algorithm designed with good security and efficient use of computing resources. The research methodology involves the development and testing of a TOTP-based two-factor authentication system with the HMAC algorithm using the PHOTON hash function. The results of the study show that the system is able to withstand brute-force attacks and birthday attacks. In addition, the TOTP function implementing PHOTON has a faster execution time than SHA-1 and SHA-2.

Keywords: *Two-factor authentication, TOTP, PHOTON hash, HMAC*

1. PENDAHULUAN

Autentikasi adalah proses memverifikasi atau memastikan keaslian suatu entitas. Autentikasi sangat penting untuk melindungi data sensitif. Metode autentikasi yang paling sering digunakan adalah kata sandi. Ini terjadi karena metode kata sandi bersifat sederhana, mudah dikelola, dan hemat biaya (Barkadehi et al., 2018). Namun, dalam banyak kasus, terjadi praktik buruk dalam implementasi metode kata sandi terutama dari sisi pengguna yang menyebabkan kerentanan terhadap peretasan. (Ezugwu et al., 2023). Salah satu metode untuk memitigasi kerentanan ini adalah dengan menerapkan dua faktor autentikasi. Metode ini dapat meningkatkan keamanan sistem dengan cara mengombinasikan metode password dengan bentuk identifikasi lain seperti faktor kepemilikan pribadi yang dapat mencakup token yang bersifat sekali pakai (Papathanasaki et al., 2022).

Time-based One-Time Password (TOTP) merupakan sebuah mekanisme autentikasi dua faktor

dengan cara men-*generate One-Time Password* (OTP) berdasarkan *secret key* bersama dan waktu (M'Raihi et al., 2011). Algoritma TOTP menggunakan sebuah fungsi hash untuk menghasilkan OTP. SHA-1 menjadi standar fungsi hash yang digunakan dalam TOTP (M'Raihi et al., 2011). Namun, seiring berjalannya waktu, banyak kekhawatiran terhadap keamanan SHA-1. Penelitian ini mencoba untuk meningkatkan keamanan TOTP dengan mengubah fungsi *hash* yang digunakan. Dari SHA-1 menjadi PHOTON, fungsi *hash* ringan yang tetap memperhatikan aspek keamanan (Guo et al., 2011). PHOTON memiliki kelebihan jika dibandingkan SHA-1, yaitu keamanan dan kinerja yang lebih baik. Penelitian ini dilakukan dengan mengimplementasikan sistem autentikasi yang menggunakan algoritma TOTP dengan fungsi *hash* PHOTON dan mengujinya dari aspek keamanan dan kinerja. Penelitian ini diharapkan dapat berkontribusi untuk meningkatkan keamanan dalam dunia digital.

2. LANDASAN KEPUSTAKAAN

2.1. Autentikasi

Autentikasi adalah proses untuk memverifikasi pengguna (Zulkarnain et al., 2013). Proses autentikasi dapat melibatkan tiga entitas yang berbeda (Zulkarnain et al., 2013).

1. *Claimant*, entitas yang melakukan autentikasi ke dalam sistem dengan tujuan untuk menggunakan layanan atau mengakses data.
2. Monitor, yaitu entitas yang menyediakan layanan autentikasi dengan memastikan identitas dari *Claimant* dan memberikan izin untuk menggunakan layanan atau mengakses data.
3. *Information System* (IS), yaitu entitas yang menyediakan layanan dan akan memastikan *Claimant* menggunakan layanannya jika monitor mengautentikasinya dengan benar.

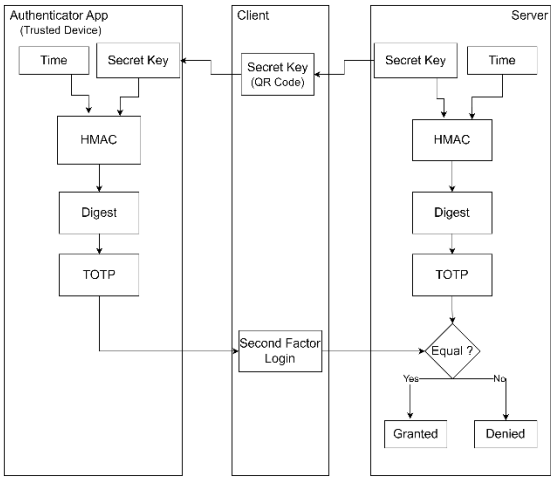
2.2. Autentikasi Dua Faktor

Autentikasi dua faktor atau *Two-Factor Authentication* (2FA) adalah proses keamanan yang mengharuskan pengguna menyediakan dua faktor autentikasi yang berbeda untuk memverifikasi identitas mereka (Vijayachandran et al., 2016). 2FA umumnya mengombinasikan kata sandi yang merupakan *something you know* dengan elemen lain seperti *One-Time Password* (OTP) yang merupakan *something you have* atau *fingerprint* yang merupakan *something you are* (Bhanderi et al., 2023).

2.3. *Time-based One-Time Password*

Time-based One-time Password (TOTP) adalah salah satu algoritma OTP (M'Raihi et al., 2011) yang proses *generating*-nya menggunakan variabel waktu sehingga OTP akan berubah-ubah sesuai waktu TOTP di-*generate*. Masa aktif OTP sesuai standar adalah 30 detik yang menyeimbangkan aspek keamanan dan kemudahan pengguna (M'Raihi et al., 2011). Dalam autentikasi dua faktor, TOTP dapat menjadi faktor kedua yang dilakukan setelah kata sandi terverifikasi.

Implementasi TOTP memerlukan perangkat sebagai *authenticator* untuk setiap pengguna. *Smartphone* dapat menjadi *authenticator* untuk autentikasi TOTP. Dengan adanya fitur kamera, proses setup TOTP menjadi mudah, server menampilkan QR Code yang menunjukkan *secret key* untuk dipindai dan disimpan dalam aplikasi *authenticator* sehingga dapat meng-*generate* kode TOTP.



Gambar 1. Autentikasi TOTP

Gambar 1 menampilkan alur kerja dari proses autentikasi TOTP. Algoritma TOTP memiliki dua *input*, yaitu *secret key* dan T. *secret key* merupakan data yang unik untuk setiap pengguna. Data ini terenkripsi dan tersimpan di *database* server. Saat autentikasi dua faktor diaktifkan, terjadi pembagian *secret key* dari server ke aplikasi *authenticator* pengguna. Sehingga *secret key* juga tersimpan dalam aplikasi *authenticator* pengguna.

$$T = floor(Unix\ timestamp/Time\ step) \quad (1)$$

Persamaan 2.1 menunjukkan operasi untuk menghasilkan variabel T. *Unix timestamp* merupakan total detik yang dimulai dari 1 Januari 1970 UTC. *Time step* merupakan masa berlaku kode TOTP dalam satuan detik. T didapatkan dari pembulatan ke bawah hasil bagi *Unix timestamp* oleh *time step*. Hal ini membuat T akan selalu sama tiap *Time step* detik.

Fungsi TOTP menggunakan algoritma kriptografi *Hash-based Message Authentication* Code (HMAC). Algoritma ini membutuhkan tiga parameter yaitu *message*, *secret key*, dan fungsi *hash*. Dalam TOTP, *message* diturunkan dari variabel waktu. *Output* dari fungsi HMAC akan di-*truncate* untuk menghasilkan 6 digit kode TOTP. Fungsi tersebut dijalankan di *authenticator* pengguna dan aplikasi server saat pengguna hendak melakukan *login* faktor kedua. Jika kode TOTP yang dimasukkan pengguna sama dengan TOTP yang dihasilkan server

maka pengguna berhasil terverifikasi dan akses diberikan. Saat melakukan autentikasi tidak ada pengiriman kode TOTP kepada pengguna, sehingga *authenticator app* bisa digunakan tanpa jaringan internet asalkan *unix timestamp authenticator app* sinkron dengan server.

2.4 Fungsi *Hash*

Fungsi *hash* ditujukan untuk mentransformasi input data dengan panjang berapa pun ke dalam output berukuran tetap, output ini disebut sebagai *hash value*, *hash code*, atau *digest* (Basar, 2011; Mihailescu dan Nita, 2021). Fungsi hash sangat penting untuk keamanan jaringan dan verifikasi integritas data dalam komunikasi dan digital *signing* (Dsilva dan Shetty, 2023). Dalam proses *hash*, *input* yang sama akan selalu menghasilkan *output* yang sama. Berapa pun panjang *input*, panjang *output* akan selalu tetap sesuai dengan algoritma *hash* yang digunakan. Perbedaan yang kecil pada *input* umumnya akan menyebabkan perubahan yang besar terhadap *output*.

2.5. *Hash-based Message Authentication Code*

Algoritma *Hash-based Message Authentication Code* (HMAC) adalah algoritma yang berfungsi untuk mengautentikasi pesan menggunakan fungsi *hash* (Krawczyk et al., 1997). HMAC bisa menggunakan fungsi *hash* iteratif apa pun yang dikombinasikan dengan *secret key* (Krawczyk et al., 1997). Tingkat keamanan HMAC bergantung pada kualitas dan ukuran *secret key*, ukuran *digest output*, dan sifat-sifat fungsi *hash* yang digunakan (Krawczyk et al., 1997).

Algoritma HMAC memiliki dua *input* yaitu *secret key* dan *message*. Dalam algoritma HMAC, didefinisikan *block bize* yang nilainya bergantung pada algoritma *hash* yang digunakan. *Block size* merupakan ukuran potongan *input* yang diproses dalam satu kali iterasi pada suatu fungsi *hash*. *Block size* menjadi ukuran dari *outer key pad* dan *inner key pad*, dua variabel penting dalam algoritma HMAC.

$$key = \begin{cases} hash(key)\ jika\ panjang\ key > B \\ pad(key, B)\ jika\ panjang\ key \leq B \end{cases} \quad (2)$$

Persamaan (2) Menunjukkan preproses untuk *key* pada algoritma HMAC. *B* adalah *block size*. Jika panjang *key* lebih dari *block size* maka *key* di-*hash* terlebih dahulu, *block size byte* pertama dari *key* menjadi *key* yang baru. Jika panjang *key* kurang dari atau sama dengan *block size* maka *key* di-*padding* dengan menambahkan bit "0" hingga panjangnya sama dengan *block size*.

$$opad = \{0x5C\}_i\ untuk\ i = 1, 2, \ldots, B \quad (3)$$

Persamaan (3) menunjukkan proses pendefinisian variabel *opad*. *Opad* merupakan *array* yang berisi nilai 0x5C sebanyak *block size*.

$$ipad = \{0x36\}_i\ untuk\ i = 1, 2, \ldots, B \quad (4)$$

Persamaan (4) menunjukkan proses pendefinisian variabel *ipad*. *Ipad* merupakan *array* yang berisi nilai 0x36 sebanyak *block size*. *opad* dan *ipad* digunakan untuk menghasilkan *outer key pad* dan *inner key pad*. Dua variabel yang akan digunakan sebagai bagian dari *input* data algoritma *hash* dalam HMAC.

$$oKeyPad = key \oplus opad \quad (5)$$

Persamaan (5) menunjukkan pendefinisian *outer key pad* pada algoritma HMAC, di mana *outer key pad* dihasilkan dari operasi XOR antara *key* dengan sebuah *array* yang berisi 0x5C sebanyak *block size*.

$$iKeyPad = key \oplus ipad \quad (6)$$

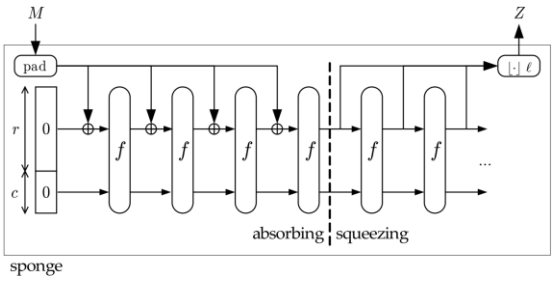
Persamaan (6) menunjukkan pendefinisian *inner key pad* pada algoritma HMAC, di mana *inner key pad* dihasilkan dari operasi XOR antara *key* dengan sebuah *array* yang berisi 0x36 sebanyak *block size*. Dari kedua variabel tersebut dihasilkan *digest* HMAC melalui persamaan berikut.

$$digest = h(oKeyPad \parallel h(iKeyPad \parallel msg)) \quad (7)$$

Persamaan (7) menunjukkan operasi untuk menghasilkan HMAC *digest*. h adalah suatu fungsi hash. *inner key pad* di-*concenate* dengan *message*. Hasil *concenation* tersebut di-*hash* untuk menghasilkan *digest* pertama. Selanjutnya, *outer key pad* di-*concenate* dengan *digest* tersebut. Hasil *concenation* di-*hash* untuk menghasilkan HMAC *digest*.

2.6. Algoritma PHOTON

PHOTON adalah fungsi *hash* yang didesain untuk perangkat yang ringan dengan sumber daya minimal (Guo et al., 2011). Fungsi ini dioptimalkan untuk mengefisiensikan penggunaan sumber daya komputasi dan memori tetapi tetap menyediakan keamanan yang baik (Guo et al., 2011). Fungsi PHOTON didesain berdasarkan konstruksi *sponge*.



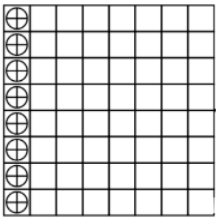
Gambar 2. Konstruksi Sponge
Sumber: Guido et al., 2011

Gambar 2 menampilkan konstruksi *sponge* yang menjadi desain dari algoritma PHOTON. Terdapat beberapa elemen penting dalam konstruksi *sponge* yaitu *State*, *message*, dan fungsi permutasi. State memiliki nilai awal yang telah terdefinisi, nilainya

akan terus berubah dipengaruhi oleh *message* dan fungsi permutasi. *Message* adalah *input* fungsi *hash* atau *plaintext*. Sedangkan fungsi permutasi atau fungsi transformasi merupakan fungsi yang berpengaruh besar terhadap kekuatan algoritma *hash*. Fungsi ini menjadi pembeda antara fungsi-fungsi *hash* yang dirancang berdasarkan *sponge construction*. *Bitrate* (*r*) merupakan panjang satu *block message* yang diproses tiap sekali iterasi. Oleh karena itu, *message* perlu di-*padding* sehingga panjangnya habis dibagi *r*.

Konstruksi *sponge* dapat dibagi menjadi dua fase yaitu *absorption* dan *squeeze*. Pada fase *absorption*, *r*-bit pertama State di-XOR dengan *r*-bit *block message* hingga semua *block message* telah diproses. Setiap satu kali iterasi, fungsi permutasi dijalankan untuk mengubah State. Setelah itu, pada fase *squeeze*, *r'*-bit pertama State akan diambil untuk menjadi *digest* hingga panjang *digest* mencapai yang didefinisikan. Seperti halnya fase *absorption*, setiap satu iterasi, fungsi permutasi dijalankan pada untuk mengubah State. Pada algoritma PHOTON, State atau *internal state* merupakan matriks dua dimensi. Tahapan dari algoritma PHOTON adalah sebagai berikut.

1. *Initialization*: Algoritma menginisiasi *state*. Dalam algoritma PHOTON, *initial state* dideklarasikan dalam bentuk konstanta *initial vector* (IV) sesuai dengan varian PHOTON yang diimplementasikan.
2. *Absorption* : Input dibagi menjadi blok-blok dengan ukuran masing-masing blok adalah *r* bits. *r* bits pertama dari internal *state* di-*XOR* dengan blok pertama untuk menghasilkan internal *state* yang baru. Kemudian dilakukan fungsi permutasi terhadap state sehingga didapatkan internal *state* yang baru lagi. Lalu diulangi untuk blok kedua dan seterusnya. Fungsi permutasi dalam PHOTON terdiri dari 4 tahap yaitu:
    a. *AddConstant*



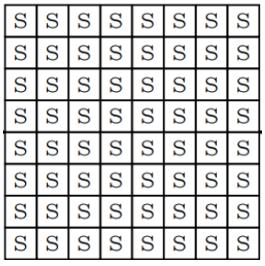
Gambar 3. Fungsi *AddConstant*
Sumber: Guo et al., 2011

Gambar 3 menunjukkan perubahan pada matriks internal *state* ketika dijalankan fungsi *AddConstant*. Fungsi ini mengubah kolom pertama matriks.

$$S'[i,0] = S[i,0] \oplus RC(v) \oplus IC(i) \quad (8)$$

Persamaan (8) merupakan fungsi pada tahap *AddConstant* dalam permutasi PHOTON. *S* adalah matriks *internal state*. *RC(v)* adalah *round constant*. *IC* adalah adalah *distinct internal constant* yang nilainya bergantung pada varian PHOTON yang digunakan.

b. *SubCells*



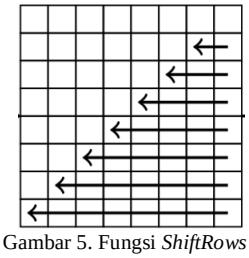
Gambar 4. Fungsi *SubCells*
Sumber: Guo et al., 2011

Gambar 4 menunjukkan perubahan pada matriks internal *state* ketika dijalankan fungsi *AddConstant*. Fungsi ini mengubah setiap nilai di dalam matriks.

$$S'[i,j] = SBOX(S[i,j]) \quad (9)$$

Persamaan (9) merupakan operasi dalam fungsi *SubCells* untuk mengganti setiap elemen dengan nilai yang baru menggunakan AES Sbox untuk varian 256-bit dan PRESENT Sbox untuk varian lain.

c. *ShiftRows*



Gambar 5. Fungsi *ShiftRows*
Sumber: Guo et al., 2011

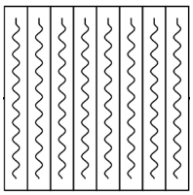
Gambar 5 menunjukkan perubahan pada matriks internal *state* ketika fungsi *ShiftRows* dijalankan. Fungsi ini menggeser secara sirkular baris kedua ke kiri sekali, baris ketiga ke kiri dua kali, baris keempat ke kiri tiga kali, dan seterusnya.

$$S'[i,j] = S[i,(j+i) \bmod d] \quad (10)$$

Persamaan (10) merupakan operasi *SubCells*. Untuk setiap baris indeks *i*, semua sel digeser ke kiri

sebanyak *i* posisi. S adalah matriks *internal state*. *d* adalah dimensi matriks internal *state* yang bergantung pada varian PHOTON yang digunakan

d. *MixColumnsSerial*



Gambar 6. Fungsi *MixColumnsSerial*
Sumber: Guo et al., 2011

Gambar 6 perubahan pada matriks internal *state* ketika fungsi *MixColumnsSerial* dijalankan. Fungsi ini mengubah setiap kolom *matrix* internal *state*.

$$(S'[0,j],\ldots,S'[d-1,j])^T = A_t^d \times (S[0,j],\ldots,S[d-1,j])^T \quad (11)$$

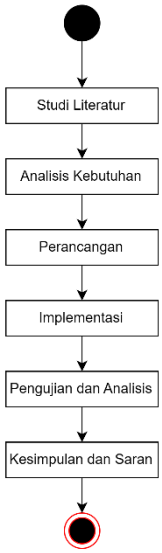
Pada persamaan (11) untuk setiap kolom j *input vector (S[0, j], . . . , S[d − 1, j])*^{T} dilakukan d kali matriks A_t = *serial*(Z_0 , . . ., Z_{d-1}). *A* adalah *mixing matrix* yang nilainya bergantung pada varian PHOTON.

3. *Squeeze*: *r'* bit pertama dari internal *state* diambil untuk menjadi *output*, lalu dilakukan fungsi permutasi terhadap internal *state* untuk menghasilkan internal *state* yang baru, setelah itu ditambahkan *r'* bits pertama dari internal *state* tersebut untuk di-*concenate* ke *output* yang diperoleh dari iterasi sebelumnya, diulangi terus sehingga *output* mencapai panjang *n*.

2.7. *Authenticator Application*

Authenticator application adalah aplikasi *mobile* yang meng-*generate* OTP (Ozkan & Bicakci, 2020). OTP ini digunakan untuk memverifikasi pengguna yang melakukan *log in* ke suatu layanan yang menerapkan autentikasi dua faktor (Ozkan & Bicakci, 2020). Di dalam *authenticator app*, dilakukan proses setup dan *code generation*. Ketika pengguna mengaktifkan autentikasi dua faktor, server akan memberikan *secret key* dalam bentuk QR code atau *string* kepada pengguna. Pengguna bisa memindai QR code sehingga didapatkan *string secret key*, dari *secret key* tersebut dilakukan *code generation* yang menghasilkan 6 digit TOTP. Karena *secret key* disimpan dalam perangkat pengguna secara *offline*, maka keamanan data dari perangkat *authenticator app* tersebut menjadi perhatian utama dalam perancangan *authenticator app*.

3. METODOLOGI PENELITIAN

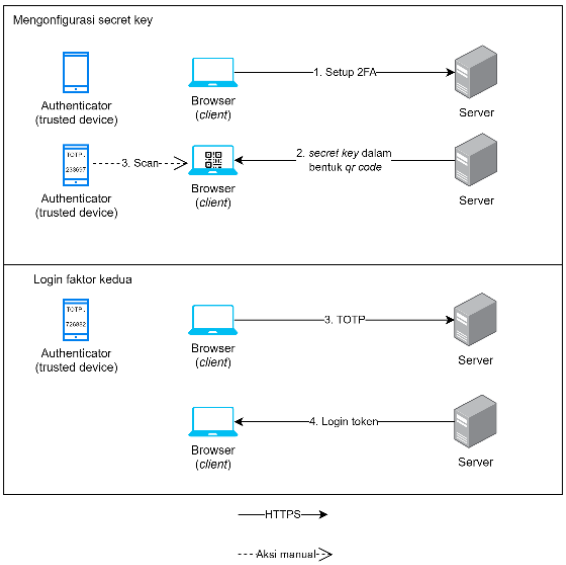


Gambar 7 Diagram Alir Metode Penelitian

Gambar 7 menampilkan alur penelitian, dimulai dari studi literatur, analisis kebutuhan, perancangan sistem, implementasi sistem, pengujian dan analisis, hingga menarik kesimpulan dan membuat saran.

4. PERANCANGAN

4.1. Gambaran Umum Sistem

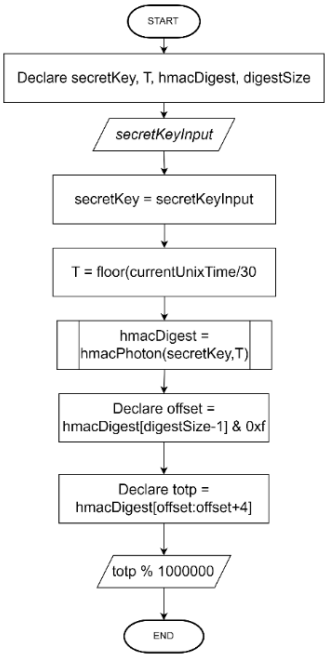


Gambar 8. Gambaran Umum Sistem

Gambar 8 menampilkan gambaran umum sistem yang hendak dikembangkan. Saat melakukan setup 2FA, pengguna menggunakan *authenticator app* untuk memindai QR *code secret key* yang diberikan server dan ditampilkan pada web *browser*. Pada fase ini *authenticator app* menyimpan *secret key*

dan meng-*generate* kode TOTP yang bisa dimasukkan pengguna setiap kali pengguna melakukan *login* faktor kedua.

4.2. *Flowchart* Algoritma TOTP



Gambar 9. *Flowchart* Algoritma TOTP

Gambar 9 menampilkan *flowchart* fungsi TOTP. Dalam implementasi peneliti, fungsi TOTP memiliki satu *input* yaitu *secret key*, sedangkan variabel T dideklarasikan di dalam fungsi. Kedua variabel tersebut menjadi *input* fungsi *hmacPhoton* yang dipanggil di dalam fungsi TOTP. *Output* dari fungsi ini disebut variabel *hmacDigest*. 4-bit terakhir dari *hmacDigest* menjadi offset untuk menentukan indeks 4-byte dari *digest* yang digunakan sebagai kode TOTP. 4-byte tersebut di-*concenate* menjadi satu bilangan integer lalu diambil 6 digit desimal terakhir untuk menjadi kode TOTP.

5. IMPLEMENTASI

5.1. Implementasi Algoritma TOTP

Algoritma 1 Fungsi getTOTP
Input: array secret key
Output: 6 digit desimal TOTP
function getTOTP(secretKey)
  TIMESTEP = 30
  T = current unix time
  T = floor(T / TIMESTEP)
  msg = encode T into 8-byte array
  digest = hmacPhoton(secretKey, msg)
  offset = digest[DIGESTBYTESIZE - 1] & 0xf
  totp = ((digest [offset] & 0x7f) << 24) |
    ((digest [offset + 1] & 0xff) << 16) |
    ((digest [offset + 2] & 0xff) << 8) |
    (digest[offset + 3] & 0xff)
  return totp % 1000000
end function

Algoritma 1 menampilkan implementasi algoritma TOTP. Fungsi *floor* membulatkan ke bawah nilai *input*. Variabel *msg* diturunkan dari variabel T dengan mengonversinya ke dalam *array* 8-byte.

Algoritma 2 Fungsi HMAC-PHOTON
Input: array secret key, array message
Output: array digest
function hmacPhoton(key, msg)
  if (length(key) > B) then
    key = Photon(key)
  else then
    key = key ‖ zeroes(B - length(key))
  end if
  outer_pad = [0x5c * B] ⊕ key
  inner _pad = [0x36 * B] ⊕ key
  return Photon(oKeyPad ‖ Photon(iKeyPad ‖ msg))
end function

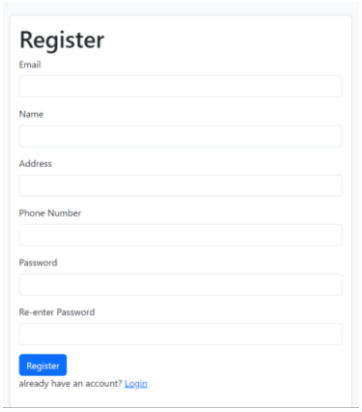
Algoritma 2 menampilkan implementasi dari algoritma HMAC-PHOTON. *B* merupakan *Block Size*. Fungsi *zeroes* mengembalikan *array* 0 sepanjang *input*.

Algoritma 3 Fungsi PHOTON
Input: array plaintext
Output: array digest
function Photon(plain):
  padded_plain = pad(plain)
  for each block **in** plain_blocks
    state = state ⊕ split(padded_plain, RATE)
    state = permutation(state)
  end for
  define output
  while length(output) < output_length
    output = output ‖ trunc(state, RATTE)
    state = permutation(state)
  end while
  return trunc(output, output_length)
end function

Algoritma 3 menampilkan implementasi dari algoritma PHOTON. Fungsi *split* mengembalikan RATE-bit (r) pertama dari *padded_plain* dan menghapusnya. Fungsi *trunc* mengembalikan RATTE-bit (r') pertama dari *state*.

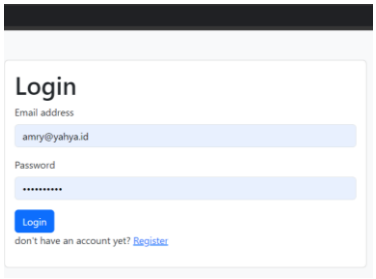
5.2. Tampilan Implementasi Sistem

Hasil dari implementasi sistem aplikasi web dan *authenticator app* ditampilkan dalam gambar-gambar berikut:



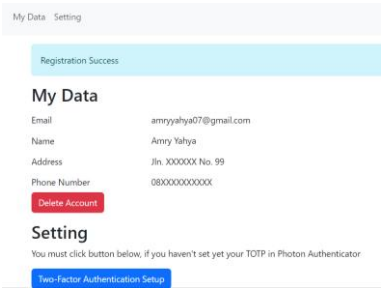
Gambar 10. Halaman Registrasi

Gambar 10 merupakan tampilan halaman registrasi. Halaman ini berisi *form* untuk membuat akun baru.



Gambar 11. Halaman *Login*

Gambar 11 merupakan tampilan halaman *login.* Halaman ini berisi *form* untuk melakukan autentikasi.

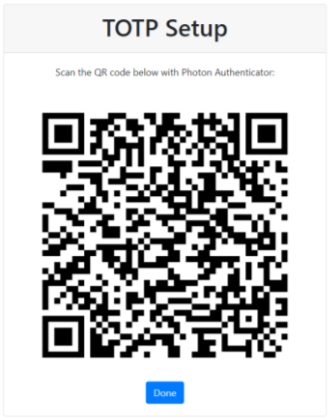


Gambar 12. Halaman Dasbor

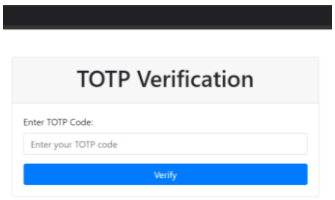
Gambar 12 merupakan tampilan halaman dasbor. Halaman ini menampilkan data-data pribadi pengguna. Halaman ini hanya boleh diakses oleh pengguna yang bersangkutan.

Gambar 13 merupakan tampilan halaman setup TOTP. Halaman ini ditampilkan ketika pengguna mengeklik tombol 'Two-Factor Authentication Setup' pada dasbor. Di halaman ini pengguna memindai *secret key* menggunakan *authenticator app*.

Gambar 14 merupakan tampilan halaman *input* TOTP. Halaman ini ditampilkan setelah pengguna memasukkan email dan *password* yang benar pada halaman *login*. Halaman ini juga ditampilkan saat setup autentikasi dua faktor untuk memastikan bahwa pengguna telah berhasil menyetup TOTP pada *authenticator app*.



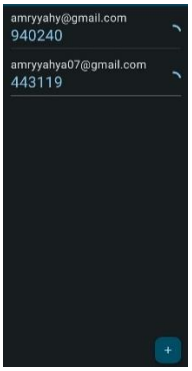
Gambar 13. Halaman Setup TOTP



Gambar 14. Halaman *Input* TOTP

Gambar 15. Tampilan *Authenticator* memindai QR *Code*

Gambar 15 menampilkan proses memindai QR Code pada *authenticator app*. Proses menyimpan *secret key* di *authenticator app.*



Gambar 16. Tampilan *Authenticator* TOTP

Gambar 16 menampilkan daftar kode TOTP yang telah disetup sebelumnya pada beberapa akun. Pada bagian kanan terdapat *loading* bar yang merepresentasikan masa aktif TOTP.

6. PENGUJIAN

6.1. Pengujian *Brute-force Attack*

Pengujian terhadap *brute-force attack* digunakan untuk menganalisis keamanan jika terjadi data *breach* terhadap email dan kata sandi pengguna. Karena TOTP berisi 6 digit angka, maka jumlah kode TOTP adalah 10^6 kemungkinan. Dalam pengujian yang dilakukan, spesifikasi perangkat adalah sebagai berikut.

- Server : e2-micro AWS Instance
  OS : Ubuntu 23.04
  Prosesor : 1 vCPU Intel Xeon Family
  RAM : 1 GiB
- Klien : Acer Swift 3 SF314-41
  OS : Linux Mint 21.2
  Prosesor : AMD Ryzen™ 5 3500U
  RAM : 8 GB
- *Tools* : BurpSuite
  Versi : Community 2023.11.1.3

Pengujian dilakukan dengan menyiapkan *file* *.txt* yang berisi semua kemungkinan kode TOTP yaitu "000000" sampai dengan "999999". *File* tersebut dimasukkan untuk menjadi *payload intruder*. Penetrasi ini dilakukan selama 30 menit. Hasilnya penetrasi gagal.

6.2. Pengujian *Birthday Attack*

Pengujian terhadap *birthday attack* digunakan untuk menganalisis keamanan jika terjadi *phising* terhadap email dan kata sandi, serta TOTP yang dikirim pengguna pada waktu tertentu. Berdasarkan teori *birthday paradox*, jika panjang digest suatu fungsi hash adalah n, terdapat 50% peluang ditemukan collision dalam $2^{n/2}$ percobaan.

Pengujian ini diimplementasikan dengan menggunakan *loop*, dalam *loop* ini, disusun *string* yang memungkinkan untuk menjadi *secret key* dengan panjang 32 karakter. TOTP di waktu *specific_time* di-*generate* dengan *string* tersebut sebagai *secret key*. Jika dihasilkan TOTP yang sama, akan dilakukan *login* dengan meng-*generate* TOTP sesuai waktu saat hendak melakukan *login.* Hal tersebut perlu dilakukan karena bisa saja *digest* yang berbeda menghasilkan TOTP yang sama. Jika berhasil, maka bisa dikatakan penetrasi berhasil karena dapat terjadi akses ke dalam sistem. *Loop* dilakukan selama 1 jam. Hasilnya penetrasi gagal.

6.3. Pengujian Waktu Eksekusi

Dalam penelitian ini, dilakukan pengukuran waktu eksekusi dari algoritma TOTP menggunakan beberapa fungsi *hash* yaitu PHOTON (160-bit, 224-bit, dan 256-bit), SHA-1 (160-bit), SHA-2 (256-bit dan 512-bit. Pengujian dilakukan sebanyak 31 kali untuk masing-masing fungsi *hash* dengan rata-rata sebagai berikut.

Tabel 1. Rata-rata Waktu Eksekusi Fungsi TOTP

Algoritma	Ukuran Digest	Rata-rata Waktu Eksekusi (detik)
PHOTON	160-bit	0.008
	256-bit	0.014
SHA-1	160-bit	0.089
SHA-2	256-bit	0.090

Pada Tabel 1 ditampilkan rata-rata waktu eksekusi dari algoritma TOTP menggunakan PHOTON, SHA-1, dan SHA-2. Hasilnya, fungsi *hash* PHOTON memiliki waktu eksekusi tercepat dibanding SHA-1 serta SHA-2.

KESIMPULAN

Penelitian ini menghasilkan sistem berupa aplikasi web dan *authenticator app* yang menerapkan algoritma TOTP dengan fungsi *hash* PHOTON. Dalam uji keamanan, sistem mampu bertahan dari *brute-force* attack dan *birthday attack*. Selain itu, kinerja sistem dengan hash PHOTON terbukti lebih baik dari SHA-1 dan SHA-2. Dari 31 kali percobaan dengan berbagai ukuran *secret key*, PHOTON lebih cepat dari SHA-1 maupun semua varian SHA-2.

DAFTAR PUSTAKA

BARKADEHI, M.H., NILASHI, M., IBRAHIM, O., ZAKERI FARDI, A. and SAMAD, S., 2018. Authentication systems: A literature review and classification. Telematics and Informatics, 35(5), pp.1491–1511. Tersedia di:<https://doi.org/10.1016/J.TELE.2018.03.018> [Diakses 5 Desember 2024].

BASAR, M.S., 2011. Summarizing data for secure transaction: A hash algorithm. African Journal of Business Management, 5(34), p.13211.

BHANDERI, D., KAVATHIYA, M., BHUT, T., KAUR, H. and MEHTA, M., 2023. Impact of Two-Factor Authentication on User Convenience and Security. In: 2023 10th International Conference on Computing for Sustainable Global Development (INDIACom). IEEE. pp.617–622.

DSILVA, R. & SHETTY, S., 2023. Applications, attacks, and advancements in cryptography and network security hash functions: a review. International Research Journal of Modernization in Engineering Technology and Science, 5(5). Tersedia di: <https://www.doi.org/10.56726/IRJMETS3937> [Diakses 22 September 2024].

EZUGWU, A., UKWANDU, E., UGWU, C., EZEMA, M., OLEBARA, C., NDUNAGU, J., OFUSORI, L. and OME, U., 2023. Password-based authentication and the experiences of end users. Scientific African, 21, p.e01743. Tersedia di: <https://doi.org/10.1016/J.SCIAF.2023.E01743> [Diakses 20 November 2023].

GUIDO, B., JOAN, D., MICHAËL, P. and GILLES, V.A., 2011. Cryptographic sponge functions.

GUO, J., PEYRIN, T. and POSCHMANN, A., 2011. The PHOTON family of lightweight hash functions. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), [online] 6841 LNCS, pp.222–239. Tersedia di: <https://doi.org/10.1007/978-3-642-22792-9_13/COVER> [Diakses 31 Agustus 2023]

KRAWCZYK, H., BELLARE, M. and CANETTI, R., 1997. RFC2104: HMAC: Keyed-hashing for message authentication.

M'RAIHI, D., MACHANI, S., PEI, M. and RYDELL, J., 2011. TOTP: Time-Based One-Time Password Algorithm. [online] Tersedia di: <https://doi.org/10.17487/RFC6238> [Diakses 31 Agustus 2023]

MIHAILESCU, M.I., NITA, S.L., 2021. Hash functions. Cryptography and Cryptanalysis in MATLAB: Creating and Programming Advanced Algorithms, pp.83-102.

OZKAN, C. and BICAKCI, K., 2020. Security analysis of mobile authenticator applications. In: 2020 International Conference on Information Security and Cryptology (ISCTURKEY). IEEE. pp.18–30.

PAPATHANASAKI, M., MAGLARAS, L., & AYRES, N., 2022. Modern authentication methods: A comprehensive survey. AI, Computer Science and Robotics Technology.

VIJAYACHANDRAN, A., KUMAR, K.G. and STUDENT,], 2016. Anonymous Two-Factor Authentication in Distributed Systems. International Journal of Computer Science Trends and Technology, [online] 4. Tersedia melalui: <www.ijcstjournal.org> [Diakses 1 August 2024].

ZULKARNAIN, S., IDRUS, S., CHERRIER, E., ROSENBERGER, C. and SCHWARTZMANN, J.-J., 2013. A Review on Authentication Methods. Australian Journal of Basic and Applied Sciences, [online] 7(5), pp.95–107. Tersedia di: <https://hal.science/hal-00912435> [Diakses 5 Desember 2023].

Halaman ini sengaja dikosongkan