Implementasi Algoritme *Hashing* BLAKE3 pada JSON *Web Token* (JWT) untuk Mekanisme Autentikasi Aplikasi *Web* Berbasis REST-API

Irfan Sofyana Putra
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517078@std.stei.itb.ac.id

Abstract*—REST merupakan salah satu gaya arsitektur komunikasi *client-server* pada aplikasi berbasis *web*. REST merupakan kependekan dari *representational state transfer*. REST merupakan komunikasi yang bersifat *stateless*, sehingga saat membuat sebuah *request, client* harus menyertakan semua data dan parameter secara lengkap. Secara umum, REST tidak memiliki metode khusus untuk autentikasi. Hal ini menyebabkan siapapun dengan mudah mengakses, mengubah, ataupun menghapus data. Untuk mengatasi hal tersebut dibutuhkan metode autentikasi khusus. Salah satunya adalah autentikasi dengan JSON *Web Token* atau JWT. Pada implementasinya, JWT menggunakan berbagai algoritme *hash* seperti HMAC-SHA256 yang merupakan algoritme SHA256 dan menggunakan MAC. Tahun 2011 ditemukan *pseudo collision* yang menyebabkan algoritme SHA256 tidak aman lagi. Di tahun 2020 ini, ditemukan sebuah algoritme *hash* baru yaitu algoritme BLAKE3 yang merupakan pengembangan dari algoritme Blake. BLAKE3 diyakini memiliki keamanan yang lebih baik karena dibangun dengan iterasi HAIFA dan ChaCha *stream cipher*. Selain itu BLAKE3 juga memiliki kompleksitas waktu yang lebih baik dibandingkan algoritme Blake versi sebelumnya. Makalah ini akan membahas implementasi algoritme *hash* BLAKE3 sebagai alternatif penggunaan algoritme HMAC-SHA256 pada JWT untuk mekanisme autentikasi aplikasi *web* berbasis REST-API.

Keywords*—algoritme Black3, JWT, REST API, fungsi *hash

I. PENDAHULUAN

Representational State Transfer (REST) adalah sebuah standar arsitektur komunikasi *client-server* berbasis *web* untuk komunikasi pertukaran data yang umumnya menggunakan protokol HTTP (Fielding, 2000). Dengan arsitektur REST, *server* menyediakan data yang diidentifikasi oleh sebuah URL untuk diakses atau dimodifikasi. Bentuk data yang ditukarkan pada gaya komunikasi REST biasanya merupakan JSON.

Secara umum, pada arsitektur REST, tidak ada mekanisme khusus yang mengatur sebuah autentikasi. Dengan kata lain, data yang disediakan oleh server dengan mudah dapat diakses, dimodifikasi, bahkan dihapus oleh seseorang. Untuk mencegah hal-hal yang tidak diinginkan khususnya yang berkaitan dengan keamanan dan kebijakan hak akses dari arsitektur REST diperlukan sebuah sistem autentikasi yang khusus. Salah satu yang biasa digunakan adalah sistem autentikasi dengan menggunakan JSON *Web Token* (JWT). JWT merupakan sebuah token yang berbentuk *string* dengan suatu format yang digunakan untuk melakukan autentikasi dan menjamin integritas pesan yang dikirim oleh salah satu pihak. Dengan menggunakan JWT ini, maka REST dapat digunakan dengan lebih aman.

Dalam implementasinya, JWT menggunakan sebuah algoritme *hash*. Salah satu standar yang digunakan adalah algoritme HS256 yang merupakan sebuah algoritme *hash* berbasis SHA-256. Pada tahun 2011, ditemukan sebuah serangan *preimage resistance* dan *pseudo collision* pada algoritme SHA-256. Hal ini memungkinkan JWT dengan algoritme HS256 menjadi kurang aman.

Dewasa kini banyak sekali algoritme *hash* yang terus dikembangkan. Salah satu yang terbaru di tahun 2020 adalah algoritme BLAKE3. Algoritme BLAKE3 adalah sebuah algoritme *hash* berdasarkan Bao dan Blake2 yang dibuat oleh Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn pada tanggal 9 Januari 2020. BLAKE3 merupakan pengembangan terbaru dari algoritme Blake. BLAKE3 merupakan algoritme dengan fitur-fitur yang kaya seperti XOF, KDF, PRF, dan MAC. Berbeda dengan pendahulunya, yang mana merupakan keluarga dari multivariant, algoritme BLAKE3 merupakan *Merkle tree*, sehingga algoritme tersebut mendukung parallelisme derajat tak hingga baik SIMD maupun *multhithreading*. Algoritme BLAKE3 juga memiliki kompleksitas waktu terbaik dibandingkan dengan para pendahulunya.

Hal-hal tersebut yang menjadi dasar untuk melakukan implementasi algoritme BLAKE3 pada JWT dalam mekanisme autentikasi aplikasi *web* berbasis REST API. Harapannya, algoritme BLAKE3 ini dapat digunakan sebagai alternatif dalam penggunaan JWT. Makalah ini berfokus pada pembuatan sistem autentikasi sederhana berbasis dengan JWT dan algoritme BLAKE3. Sistem autentikasi akan dibuat dalam Bahasa pemrograman Python.

II. DASAR TEORI

A. Kriptografi

Kriptografi secara etimologi berasal dari Bahasa Yunani yaitu “kriptos” yang artinya “yang tersembunyi”

dan “graphein” yang artinya adalah tulisan (Prayudi, 2005). Awal mulanya kriptografi dipahami sebagai ilmu tentang menyembunyikan pesan (Sadikin, 2012), tetapi seiring perkembangan zaman hingga saat ini pengertian kriptografi berkembang menjadi ilmu tentang teknik matematis yang digunakan untuk menyelesaikan persoalan keamanan berupa privasi dan otentikasi (Diffie, 1976).

Kriptografi adalah ilmu mengenai teknik enkripsi dimana “naskah asli” (*plaintext*) diacak menggunakan suatu kunci enkripsi menjadi “naskah acak yang sulit dibaca” (*ciphertext*) oleh seseorang yang tidak memiliki kunci dekripsi. Dekripsi dengan menggunakan kunci dekripsi bisa mendapatkan kembali data asli. Probabilitas mendapat kembali naskah asli oleh seseorang yang tidak mempunyai kunci dekripsi dalam waktu yang tidak terlalu lama adalah sangat kecil.

Di dalam bidang kriptografi, terdapat beberapa terminology yang sering digunakan, yaitu:

1. Pesan

   Informasi yang dapat dibaca dan dimengerti maknanya (baik persepsi secara visual maupun audial). Nama lain: plainteks, *plain-image*, *plain-video*.
2. Pengirim

   Pengirim adalah subjek atau pihak yang mengirimkan sebuah pesan. Pengirim dapat berupa orang, komputer, mesin, dan lain-lain.
3. Penerima

   Penerima adalah subjek atau pihak yang menerima pesan dari pengirim. Penerima juga dapat berupa orang, komputer, mesin dan lain-lain.
4. Cipherteks

   Cipherteks adalah pesan yang telah disandikan sehingga lebih sulit untuk dibaca dan menjadi kurang bermakna lagi. Tujuannya adalah agar pesan tidak dapat dibaca oleh pihak yang tidak berhak. Nama lainnya adalah kriptogram.
5. Enkripsi

   Enkripsi adalah proses untuk menyandikan sebuah plainteks menjadi cipherteks.
6. Dekripsi

   Dekripsi adalah proses yang berlawanan dengan proses enkripsi. Proses dekripsi adalah proses untuk mengembalikan cipherteks menjadi plainteks semula.
7. Cipher

   Cipher adalah algoritme yang digunakan untuk melakukan enkripsi maupun dekripsi. Definisi lain dari cipher adalah sebuah fungsi matematika yang digunakan untuk enkripsi dan dekripsi pesan.
8. Kunci

   Kunci adalah parameter yang digunakan saat melakukan enkripsi maupun dekripsi.
9. Penyadap

   Penyadap adalah orang atau mesin yang mencoba menangkap pesan selama ditransmisikan.
10. Kriptanalisis

    Kriptanalisis adalah ilmu dan seni untuk memecahkan cipherteks menjadi sebuah plainteks tanpa mengetahui kunci yang digunakan. Orang yang melakukan kriptanalisis disebut sebagai kriptanalis.

B. Fungsi *Hash*

Fungsi *hash* adalah salah satu algoritme yang ada pada ilmu kriptografi. Fungsi *hash* adalah sebuah fungsi yang mengkompresi sebuah pesan (M) berukuran sembarang menjadi sebuah *string* (h) yang berukuran tetap. Luaran atau *output* dari fungsi *hash* tersebut dinamakan pesan ringkas (*message-digest*) atau nilai *hash* (*hash value*).

Fungsi *hash* bersifat satu arah atau sering disebut *irreversible*, yang artinya adalah sebuah pesan yang telah dilakukan *hash* tidak bisa dikembalikan menjadi pesan semula.

Sebuah fungsi *hash* H memiliki beberapa sifat yang harus diketahui, yaitu:

1. *Collision resistance*

   Sebuah sifat dimana fungsi yang dihasilkan akan membuat sesorang sukar untuk mencari dua buah input *a* dan *b* sedemikian sehingga $H(a) = H(b)$.
2. *Preimage resistance*

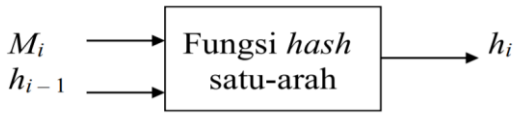
   Sebuah sifat dimana untuk sembarang *output y*, sangat menemukan sebuah *input a* sedemikian sehingga $H(a) = y$
3. *Second preimage resistance*

   Sebuah sifat dimana untuk setiap *input a* dan *output y*, sukar untuk menemukan input kedua *b* sedemikian sehingga $H(b) = y$

Masukan sebuah fungsi *hash* adalah blok pesan (*M*) dan keluaran dari *hashing* blok pesan sebelumnya,

$$h_i = H(M_i, h_{i-1})$$

Skema sebuah fungsi *hash* ditunjukkan pada Gambar di bawah:



Gambar 1. Fungsi *hash* satu-arah

Fungsi *hash* satu arah tidak tepat apabila disebut sebagai proses enkripsi, meskipun nilai *hash* tidak memiliki makna, nilai *hash* tidak dapat ditransformasikan kembali menjadi pesan semula. Alasan lainnya adalah proses *hashing* tidak menggunakan kunci. Di dalam ilmu kriptografi, terdapat beberapa fungsi yang biasa digunakan adalam proses *hashing*. Beberapa diantaranya adalah MD5, SHA256, SHA3 (*keccak*), dan BLAKE.

Fungsi *hash* memiliki beberapa manfaaat. Diantaranya adalah:

1. Menjaga Integritas Pesan

   Fungsi *hash* sangat peka terhadap perubahan 1 bit pada pesan. Oleh sebab itu, fungsi *hash* dapat digunakan untuk menjaga keaslian dan integritas dari sebuah pesan dengan baik.
2. Menghemat Waktu Pengiriman Pesan

   Seperti yang telah dijelaskan sebelumnya, fungsi *hash* adalah fungsi yang mengkompresi sebuah pesan. Oleh sebab itu, fungsi *hash* dapat digunakan untuk menghemat waktu pengiriman pesan. Salah satu

penggunannya adalah untuk keperluan memverifikasi sebuah salinan arsip dengan arsip asli.

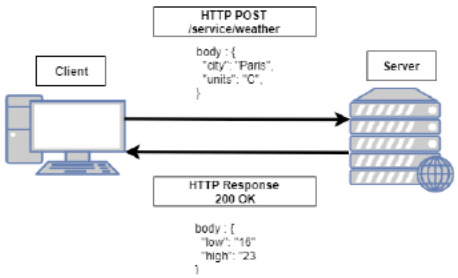
3. Menormalkan Panjang Data yang Beraneka Ragam

Fungsi *hash* dapat digunakan untuk menormalkan Panjang data yang beraneka ragam. Salah satu contoh pengunaannya adalah untuk menstandarisasi Panjang *password* pada sebuah aplikasi dengan menyimpan *password* tersebut sebagai nilai *hash*.

C. *Representational State Transfer* (REST)

Representational State Transfer (REST) adalah bagian dari *hyptertext transfer protocol* atau HTTP dimana menyediakan antarmuka yang seragam seperti membuat, mengambil, memperbarui, menghapus, dan memanipulasi sumber daya dengan pertukaran representasi. REST bersifat *stateless* yang artinya pesan tidak memiliki ketergantungan pada keadaan percakapan (*World Wide Web Consortium*, 2004).

Arsitektur REST digunakan untuk memanipulasi data pada sebuah sistem dengan menggunakan metode protokol HTTP seperti GET, POST, DELETE, dan sebagainya. Data diidentifikasi dengan sebuah *uniform resource locator* (URL) untuk digunakan sebagai antar muka dalam memanipulasi sumber daya. Dalam arsitektur REST, data yang dipertukarkan dapat berupa XML, HTML, maupun JSON. Dengan menggunakan protokol HTTP/HTTPS yang bersifat *stateless*, arsitektur REST ditujukan untuk meningkatkan performa dan juga reliabilitas dan skalabilitas.



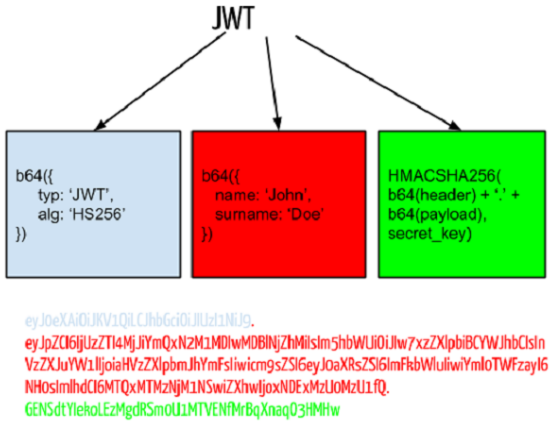
Gambar 2. Arsitektur REST *Client-Server*

(Bagus Satria et.al, 2018)

D. JSON *Web Token*

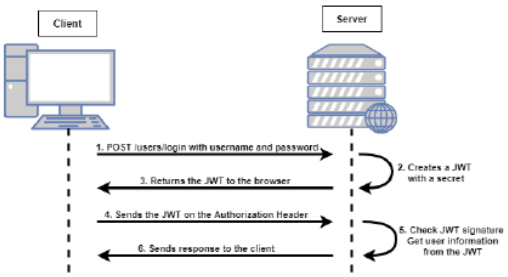
JSON *Web Token* (JWT) merupakan sebuah *access token* dengan format *Javascript Object Notation* atau biasa disebut dengan JSON (Bradley, 2015). Token tersebut akan digunakan sebagai autentikasi pada sebuah komunikasi *client-server*. Token JWT didesain dengan ringkas dan padat.

JWT adalah sebuah *token* yang terdiri dari 3 bagian yaitu *header*, *payload* dan *signature*. Bagian *header* akan berisi tentang algoritme dan jenis token yang digunakan. Bagian ini hanyalah sebuah *string* yang di-*encode* menggunakan base64. Sehingga untuk mendapatkan nilai aslinya, kita bisa menggunakan berbagai macam *tools* base64 *decoder*. Bagian kedua adalah *payload*. *Payload* ini berisi data-data yang ingin dikirim melalui sebuah token. Sama seperti bagian *header*, bagian ini juga di-*encode* dengan menggunakan base64. Dalam penerapannya untuk autentikasi, biasanya data ini merupakan data penting seorang pengguna seperti email, id/uuid, dan juga data yang berkaitan dengan otorisasi seperti role. Bagian ketiga adalah *signature*. *Signature* ini adalah sebuah nilai *hash* hasil gabungan dari *header*, *payload* dan juga sebuah *secret-key* (berupa *string* random panjang) yang harus dirahasiakan. Algoritme *hash* yang digunakan mengikuti dari apa yang sudah ditentukan di dalam *header*. *Signature* berguna untuk memverifikasi bahwa *header* maupun *payload* yang ada pada *token* tidak berubah dari nilai aslinya.



Gambar 3. Contoh JWT Token (https://code.tutsplus.com/id/tutorials/token-based-authentication-with-angularjs-nodejs--cms-22543)

Mekanisme autentikasi diawali dengan memberikan kredensial *client* kepada *server* untuk diidentifkasi kemudian apabila berhasil melakukan autentikasi maka diberikan sebuah *access token* yang seterusnya dapat digunakan untuk mengakses sumber daya yang ada pada *server* melalui *request*.



Gambar 4. Mekanisme Autentikasi dengan JWT

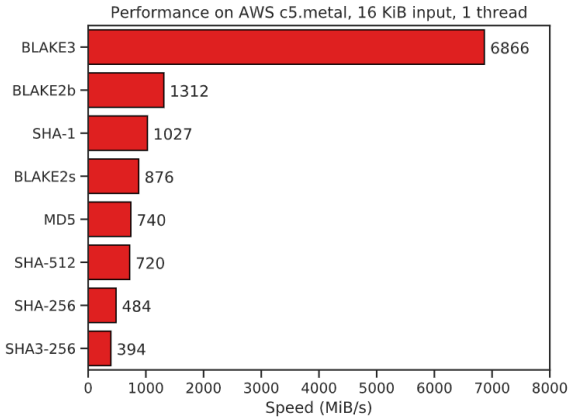
(Bagus Satria et.al, 2018)

E. Algoritme BLAKE3

BLAKE3 adalah sebuah fungsi *hash* kriptografi berdasarkan Bao dan BLAKE2, dibuat oleh Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, dan Zooko Wilcox-O'Hearn pada tanggal 9 Januari 2020 di acara *Real Word Crypto*. BLAKE3 adalah algoritme tunggal dengan banyak fitur yang diinginkan (paralelisme, XOF, KDF, PRF, dan MAC), berbeda dengan BLAKE dan BLAKE2, yang merupakan keluarga algoritme dengan banyak varian.

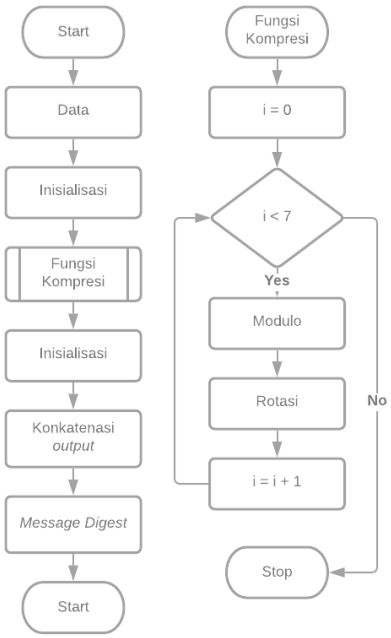
BLAKE3 adalah pohon Merkle, sehingga mendukung derajat paralelisme yang tidak terbatas (baik SIMD dan multithreading).

BLAKE3 dirancang untuk bekerja secepat mungkin. Hasil beberapa *benchmark* mengatakan bahwa BLAKE3 secara konsisten beberapa kali lebih cepat dari pendahulunya yaitu BLAKE2. Fungsi kompresi BLAKE3 sangat mirip dengan BLAKE2s, dengan perbedaan terbesar adalah jumlah putaran dikurangi dari 10 menjadi 7, perubahan berdasarkan asumsi bahwa kriptografi saat ini terlalu konservatif. Selain menyediakan paralelisme, format pohon Merkle juga memungkinkan untuk streaming terverifikasi (verifikasi *on-the-fly*) dan pembaruan tambahan.



Gambar 5. Benchmark BLAKE3 dengan Beberapa Algoritme Hash (https://github.com/BLAKE3-team/BLAKE3/)

Adapun skema global dari algoritme BLAKE3 dapat dilihat pada *flowchart* dibawah ini:



Gambar 6. Skema Global Algoritme Blake3

III. Rancangan Solusi dan Implementasi

Pada bagian ini akan disampaikan deskripsi umum solusi yang dibuat beserta rancangan solusi dan implementasinya.

A. Deskripsi Umum Solusi

Solusi yang akan dibuat adalah mengimplementasikan algoritme *hash* BLAKE3 sebagai alternatif penggunaan algoritme HMAC-SHA256 pada JWT untuk mekanisme autentikasi aplikasi *web* berbasis dengan REST-API. Solusi yang akan dibuat akan memenuhi kebutuhan fungsional berikut:

1. *Client* dapat melakukan *register* pada *server*.
2. *Client* dapat melakukan autentikasi dengan menggunakan *email* dan *password*.
3. *Server* REST akan memberikan sebuah JWT *token* apabila autentikasi berhasil.
4. *Server* REST dapat melakukan verifikasi *token* untuk setiap *request* yang dikirimkan oleh *client.*
5. *Client* dapat mengakses data informasi dengan menggunakan *token* yang diberikan oleh *server REST*.

Karena aplikasi yang dibangun akan berbasis REST-API, maka pertukaran data akan dilakukan dalam format data JSON.

B. Rancangan Solusi

Solusi yang akan dibangun akan menggunakan Bahasa pemrograman Python dengan menggunakan *framework flask*. Kakas lain yang digunakan untuk mengimplementasikan solusi adalah MySQL sebagai basis data.

Pada makalah ini dibuat 4 buah API yang dapat digunakan. Detail API yang digunakan ada pada tabel berikut:

URI	Butuh Autentikasi	Contoh *Body* (JSON)	Penjelasan
POST /register	Tidak	{ "username": "irfan", "password": "kripto", "name":"irfan sofyana "email": "isp@gmail.com" }	Membuat sebuah akun baru sesuai dengan data yang ada pada *body*
POST /login	Tidak	{ "username": "irfan", "password": "kripto" }	Login pada aplikasi dengan sebuah username dan password
GET /users/:userna	Ya	-	Mendapatkan detail

me			informasi seorang user
PUT /users/:username	Ya	{ "password": "password baru", "name": “Irfan doank” "email": "irfan@gmail.com" }	Update detail informasi seorang user

Tabel 1. Desain API

API *register* digunakan untuk menambah sebuah user ke database. Kemudian API *login* digunakan untuk melakukan autentikasi. Apabila *login* berhasil, maka *server* akan mengembalikan sebuah JWT *token*. JWT *token* inilah yang akan digunakan untuk melakukan autentikasi selanjutnya apabila *client* ingin mengirimkan *request* pada sebuah API. Pada makalah ini *user* akan memerlukan autentikasi pada API /user/:username untuk mendapatkan detail dan informasi dari seorang user berdasarkan username. API PUT /users/:username akan digunakan untuk melakukan *update* detail informasi tentang seseorang. Pada kasus ini JWT akan digunakan sebagai otorisasi sehingga user hanya dapat mengubah informasi dirinya sendiri.

Skema basis data yang digunakan cukup sederhana, yaitu hanya terdapat 1 buah tabel *users* yang terdiri dari atribut *username*, *password*, nama, dan *email*. Password disimpan pada basis data dengan menggunakan kakas *werkzeug*.

C. Implementasi Solusi

Implementasi dilakukan menjadi dua tahap. Tahap pertama adalah pembangunan fungsi JWT dengan menggunakan algoritme *hash* BLAKE3 sebagai alternatif dari penggunaan algoritme HMACSHA-256. Implementasi yang dibuat berupa membuat 2 fungsi utama yaitu fungsi untuk membangkitkan *token* dan fungsi untuk memverifikasi token.

Fungsi untuk membangkitkan JWT *token* dan melakukan verifkasi *token* dapat dilihat pada gambar berikut:

```
def create_token(payload, secret, header = json.dumps({"typ": "JWT","alg": "BLAKE3"})):
    header_encoded = base64_encode(header)
    payload_encoded = base64_encode(payload)
    signing = blake3(header_encoded + b"." + payload_encoded, key = secret.encode('ascii')).hexdigest()

    return f"{header_encoded.decode('ascii')}.{payload_encoded.decode('ascii')}.{signing}"
```

Gambar 7. Algoritme Pembangkitan *Token*

```
def verify_token(token, secret):
    try:
        [header, payload, signing] = token.split('.')
        generated_signing = blake3(
            header.encode('ascii') + b'.' + payload.encode('ascii'),
            key = secret.encode('ascii')
        ).hexdigest()

        return signing == generated_signing
    except Exception as e:
        print("Error occured when verify token", e)
        traceback.print_exc()
```

Gambar 8. Algoritme Verifikasi *Token*

Algoritme verifkasi *token* digunakan pada sebuah *middleware* yang digunakan untuk melakukan autentikasi *client* sebelum diteruskan menuju *controller*. Adapun *middleware* yang dibuat adalah sebagai berikut:

```
class middleware():
    def __init__(self, app):
        self.app = app

    def __call__(self, environ, start_response):
        request = Request(environ)
        paths = request.path.split('/')

        protected_base_uri = ['users']

        if (paths[1] in protected_base_uri):
            if ('HTTP_AUTHORIZATION' not in environ):
                resp = {"status": 401, "data": {'message': 'Unauthorized'}}
                res = Response(json.dumps(resp), mimetype= 'application/json', status=401)
                return res(environ, start_response)

            auth_headers = request.headers['Authorization'].split(' ')
            if (len(auth_headers) != 2):
                resp = {"status": 401, "data": {'message': 'Authorization key is not Bearer Token type!'}}
                res = Response(json.dumps(resp), mimetype= 'application/json', status=401)
                return res(environ, start_response)

            jwt_token = auth_headers[1]
            if (verify_token(jwt_token, os.getenv('SECRET_KEY'))):
                [header, payload] = extract_header_payload(jwt_token)
                environ['user'] = payload
```

```
                return self.app(environ, start_response
)

            resp = {"status": 401, "data": {'message':
'Token is not verified!'}}
            res = Response(json.dumps(resp), mimetype=
'application/json', status=401)
            return res(environ, start_response)

        return self.app(environ, start_response)
```

Gambar 9. Implementasi *Middleware*

Setelah tahap pertama selesai, maka dilakukan tahap kedua yaitu pembangunan API utama agar dapat melayani *request* yang dikirimkan oleh *client*. Implementasi dari 4 API yang dibuat adalah sebagai berikut:

```
@app.route('/register', methods=['POST'])
def register():
    username = request.json['username']
    password = request.json['password']
    name = request.json['name']
    email = request.json['email']

    user = find_user(username)
    if (user is not None and len(user) > 0):
        return send_response(400, {'message': 'username
 is already taken!'})

    try:
        curr = mysql.connection.cursor()
        curr.execute(
            "INSERT INTO users VALUES(%s, %s, %s, %s)",

            (username, generate_password_hash(password)
, name, email)
        )
        mysql.connection.commit()
        curr.close()

        return send_response(200, {'message': 'user cre
ated!'})
    except Exception as err:
        print(err)
        return send_response(500, {'message': 'error wh
en creating user!'})
```

Gambar 10. Implementasi *Register* API

```
@app.route('/login', methods=['POST'])
def login():
    username = request.json['username']
    password = request.json['password']

    try:
        user = find_user(username)

        if (user is None or len(user) == 0):
            return send_response(404, {'message': 'user
 not found!'})

        hashed_password = user[0][1]

        if not(check_password_hash(hashed_password, pas
sword)):
            return send_response(401, {'message': 'user
name or password is incorrect!'})

        data = json.dumps({'username': user[0][0]})
        jwt_token = create_token(payload=data, secret=o
s.getenv('SECRET_KEY'))
        return send_response(200, {'jwt_token': jwt_tok
en})
    except Exception as err:
        print(err)
        return send_response(500, {'message': 'error wh
en login!'})
```

Gambar 11. Implementasi *Login* API

```
@app.route('/users/<username>', methods=['GET'])
def get_specific_user(username):
    try:
        curr = mysql.connection.cursor()
        curr.execute('SELECT username, name, email FROM
 users WHERE username = %s', (username,))
        user = curr.fetchall()
        curr.close()

        if (user is None):
            return send_response(200, {})
        else:
            return send_response(200, {
                'username': user[0][0],
                'name': user[0][1],
                'email': user[0][2]
            })
    except Exception as err:
        print(err)
        return send_response(500, {'message': 'error wh
en GET detail of a user'})
```

Gambar 12. Implementasi GET User Detail

```python
@app.route('/users/<username>', methods=['PUT'])
def update_specific_user(username):
    try:
        user = json.loads(request.environ['user'])
        if (user['username'] != username):
            return send_response(403, {'message': 'you
are not allowed to modify other user!'})

        new_name = request.json['name']
        new_email = request.json['email']
        new_password = request.json['email']

        curr = mysql.connection.cursor()
        curr.execute(
            'UPDATE users SET name = %s, email = %s, pa
ssword = %s WHERE username = %s',
            (new_name, new_email, generate_password_has
h(new_password), username)
        )
        mysql.connection.commit()
        curr.close()

        return send_response(200, {'message': 'user inf
ormation is successfully updated!'})
    except Exception as err:
        print(err)
        return send_response(500, {'message': 'error wh
en update a user'})
```

Gambar 13. Implementasi *PUT User*

IV. PENGUJIAN

Pengujian dilakukan dengan dua tahap. Pengujian pertama adalah pengujian fungsionalitas *server* dan tahap kedua adalah pengujian waktu autentikasi.

A. Pengujian Fungsionalitas

Pengujian fungsionalitas *server* dilakukan dengan bantuan kakas *postman*. Pengujian yang dilakukan adalah sebagai berikut:

1. Pengujian *register* user

Request payload (JSON)

```
{
   "username": "if4020",
   "password": "kripto menyenangkan",
   "name": "Kriptografi",
   "email": "kripto@itb.ac.id"
}
```

Response payload (JSON)

```
{
   "data": {
      "message": "user created!"
   },
   "status": 200
}
```

2. Pengujian *login* user sukses

Request payload (JSON)

```
{
 "username": "if4020",
 "password": "kripto menyenangkan"
}
```

Response payload (JSON)

```
{
    "data": {
                "jwt_token":
          "eyJ0eXAiOiAiSldUIiwgImFsZyI6IC
          JCTEFLRTMifQ==.eyJ1c2VybmFtZ
          SI6ICJpZjQwMjAifQ==.15ed9bb300
          07499d0216b41b4f9abf07ea8472e72
          7cce1dc181f4ab14657221d"
     },
   "status": 200
}
```

3. Pengujian *login* dengan kredensial salah

Request payload (JSON)

```
{
   "username": "if4020",
   "password": "kripto sulit"
}
```

Response payload (JSON)

```
{
    "data": {
          "message":"username or password is
          incorrect!"
    },
    "status": 401
}
```

4. Pengujian *GET user* tanpa *token*

Request parameter: “if4020”

Response payload (JSON)

```
{
 "status": 401,
 "data": {
     "message": "Unauthorized"
  }
}
```

5. Pengujian *GET user* dengan *token* salah

Request parameter: “if4020”

Response payload (JSON)

```
{
   "status": 401,
    "data": {
          "message": "Token is not verified!"
     }
}
```

6. Pengujian *GET user* dengan *token* tepat
Request parameter: “if4020”

Response payload (JSON)

```
{
   "data": {
      "email": "kripto@itb.ac.id",
      "name": "Kriptografi",
      "username": "if4020"
   },
   "status": 200
}
```

7. Pengujian *PUT user* dengan username tidak sesuai

Request parameter: “if4031”

Request payload (JSON)

```
{
  "password": "ini password",
"name": "bukan matkul kripto",
"email": "pat@gmail.com"
}
```

Response payload (JSON)

```
{
    "data": {
       "message": "you are not allowed to
modify other user!"
     },
    "status": 403
}
```

8. Pengujian *PUT user* dengan token benar

Request payload (JSON)

```
{
  "password": "kripto sangat sangat
menyenangkan",
"name": "kripto matkul if",
"email": "kripto@gmail.com"
}
```

Response payload (JSON)

```
{
   "data": {
     "message": "user information is
successfully updated!"
   },
  "status": 200
}
```

B. Pengujian Waktu Autentikasi

Pengujian waktu autentikasi digunakan untuk mengetahui waktu yang digunakan saat melakukan autentikasi dengan menggunakan algoritme BLAKE3.

Prosedur pengujian dilakukan sebanyak 10 kali percobaan. Pada akhir percobaan, dihitung waktu rata-rata yang dibutuhkan dalam melakukan autentikasi. Pada percobaan ini, proses autentikasi berjalan secara sekuensial sehingga proses autentikasi dijalankan secara satu per satu. Berikut adalah hasil percobaan yang dilakukan

Percobaan ke-	Waktu
1	1.02 ms
2	1.05 ms
3	1.19 ms
4	1.24 ms
5	1.18 ms
6	0.99 ms
7	0.98 ms
8	1.31 ms
9	1.28 ms
10	0.96 ms
Rata-rata: 1.12 ms	

Tabel 2. Tabel Percobaan Waktu Autentikasi

Berdasarkan tabel tersebut, didapatkan bahwa rata-rata waktu yang diperlukan untuk melakukan autentikasi dengan menggunakan algoritme BLAKE3 adalah 1.12 ms. Dapat dikatakan waktu verikasi yang dibutuhkan sangat cepat dan efisien.

V. KESIMPULAN DAN SARAN

Algoritme BLAKE3 adalah sebuah algoritma *hash* yang dapat digunakan untuk berbagai macam hal. Salah satunya dapat digunakan sebagai fungsi untuk melakukan *hash* pada metode autentikasi dengan menggunakan JWT.

Makalah ini membahas bagaimana mengimplementasikan sebuah algoritme BLAKE3 untuk keperluan autentikasi dengan metode JWT pada aplikasi *web* berbasis REST API. Hasil yang didapatkan cukup memuaskan karena sistem autentikasi dapat digunakan dengan baik dan juga memiliki performansi waktu yang cepat dan efisien. Saat ini terdapat kekurangan pada implementasi yaitu *token* tidak pernah *expired*, sehingga hal tersebut merupakan saran pengembangan yang dapat dilakukan kedepannya.

VI. UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena berkat-Nya penulis dapat menyelesaikan makalah ini dengan baik. Penulis juga ingin mengucapkan terima kasih kepada Bapak Rinaldi Munir selaku dosen mata kuliah IF4020 Kriptografi yang telah memberikan banyak ilmu baik di dalam perkuliahan maupun di luar perkuliahan. Penulis juga tak lupa mengucapkan terima kasih kepada ibu, keluarga, serta teman-teman khususnya Asif, Juniardi, Winston, Hanif, Faiz, Abda, Jofiandy, dan Hamzah yang terus memberikan dukungan dalam menjalankan perkuliahan khususnya pada saat masa pandemi COVID-19. Terakhir, penulis memohon maaf apabila di dalam penulisan makalah ini terdapat kesalahan baik yang disengaja maupun tidak disengaja. Penulis berharap makalah ini mampu berguna bagi banyak orang.

REFERENSI

[1] Munir, Rinaldi. 2020. Slide Kuliah IF4020 Kriptografi: Pengantar Kriptografi

[2] Munir, Rinaldi. 2020. Slide Kuliah IF4020 Kriptografi: Fungsi *Hash*

[3] Satria, Bayu, et.al., 2018. Implementasi Algoritme Blake2s pada JSON Web Token (JWT) sebagai algoritme *Hashing* untuk mekanisme layanan REST-API

[4] Salma, Alfiana Irsyada. 2017. Mengenal Konsep JSON Web Token (JWT). https://www.dumetschool.com/blog/mengenal-konsep-json-web-token-jwt. Diakses pada tanggal 20 Desember 2020.

[5] O'Connor, Jack. et.al. 2020. Blake3 one function, fast everywhere. https://blake3.io. Diakses pada tanggal 20 Desember 2020.

[6] Romadhoni, Firmansyah. 2020. Perbedaan antara API, REST API, dan Restful API. https://medium.com/jagoanhosting/perbedaan-antara-api-rest-api-dan-restful-api-6a66d655a6c2. Diakses pada tanggal 19 Desember 2020.

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.

Ciamis, 20 Desember 2020

Irfan Sofyana Putra 13517078