IMPLEMENTASI ALGORITME BLAKE2S PADA JSON WEB TOKEN (JWT) SEBAGAI ALGORITME *HASHING* UNTUK MEKANISME AUTENTIKASI LAYANAN REST-API

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Bagus Satria Wiguna
145150200111202

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI ALGORITME BLAKE2S PADA JSON WEB TOKEN (JWT) SEBAGAI ALGORITME *HASHING* UNTUK MEKANISME AUTENTIKASI LAYANAN REST-API

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Bagus Satria Wiguna
NIM: 145150200111202

Skripsi ini telah diuji dan dinyatakan lulus pada
26 Juli 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T, M.sc	Widhi Yahya, S.Kom, M.T
NIK : 201102 831228 2 001	NIK: 201607 891121 1 000

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 26 Juli 2018

Bagus Satria Wiguna

NIM: 145150200111202

KATA PENGANTAR

Assalamualaikum Wr. Wb. Alhamdulillah, puji syukur kehadirat Allah SWT atas limpahan rahmat dan hidayahnya penulis dapat menyelesaikan skripsi tepat waktu. Skripsi ini disusun oleh penulis sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.

Penulis menyadari bahwa tanpa bantuan, bimbingan, serta dorongan dari semua pihak, penyelesaian skripsi ini tidak mungkin bisa terwujud. Pada kesempatan ini penulis menyampaikan rasa terima kasih sebesar-besarnya kepada:

1. Ibu Endang Budi Wachjuni dan Bapak Djoko Marsudi yang memberikan kasih sayang, dukungan dan doa yang tiada henti.
2. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D selaku Dekan Fakultas Ilmu Komputer Universtitas Brawijaya.
3. Bapak Tri Astoto Kurniawan , S.T, M.T, Ph.D selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
4. Bapak Agus Wahyu Widodo , S.T, M.Cs selaku Ketua Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya .
5. Ibu Ari Kusyanti, S.T, M.Sc selaku dosen pembimbing pertama yang telah memberikan bimbingan, saran, motivasi, dan pengarahan dalam penyusunan skripsi ini.
6. Bapak Widhi Yahya, S.Kom, M.T selaku dosen pembimbing kedua yang telah memberikan bimbingan, saran, motivasi, dan pengarahan dalam penyusunan skripsi ini.
7. Sahabat terbaik : Arsana Yudistira, Andre Rizal Sinaga, Kevin Charlie, Imam Santosa, Rizki Romadhoni,Riski Pradana, Suhhy Ramzini, Bayu Bhuana, Robihamanto, Ridho Pratama, Salma Mutia, Dyah Ayu, Rizqi Fauzi, Rizal Fahturrizqi, Ivan Yulfrian, Cahyaningtyas Sekar, Alan Maulana Hamid, Iskar Maulana dan teman-teman yang belum saya sebutkan. Terima kasih untuk waktu, semangat, dukungan dan segala bentuk bantuan yang luar biasa.
8. Teman teman Tabook Indonesia: Alvan Zaputra, Dimas Anantyo, Yuri Pratama, Alifatul Mufarohah, Arin Siska terima kasih atas dukungan dan semangat yang diberikan kepada penulis.
9. Pesan dan kesan dari peserta Praktikum Pemrograman Lanjut, Algoritma dan Struktur Data, Jaringan Komputer dan Rekayasa Perangkat Lunak yang memberikan kritik, saran dan dorongan semangat pada penulis.
10. Teman-teman seperjuangan Teknik Informatika 2014 yang keren dan luar biasa, terima kasih untuk hal-hal yang luar biasa, tetap semangat dan semoga kita bisa lebih baik lagi ke depannya.
11. Semua pihak yang telah memberikan bantuan baik secara langsung maupun tidak langsung dalam penyusunan skripsi ini.

Dalam penyusunan skripsi ini penulis menyadari bahwa skripsi ini belum sempurna karena keterbatasan ilmu dan kendala kendala lain yang terjadi selama pengerjaan skripsi ini. Semoga tulisan ini dapat bermanfaat dan dapat digunakan untuk pengembangan lebih lanjut.

Wassalamualaikum Wr. Wb.

Malang,26 Juli 2018

Penulis

145150200111202

ABSTRAK

Bagus Satria, Implementasi Algoritme BLAKE2S Pada JSON Web Token Sebagai Algoritme *Hashing* Untuk Mekanisme Autentikasi Layanan REST API.

Pembimbing: Ari Kusyanti, S.T, M.sc dan Widhi Yahya, S.Kom, M.T

REST merupakan arsitektur komunikasi *client-server* berbasis *web* untuk komunikasi data yang menggunakan protokol HTTP. Pada arsitektur REST, REST *server* menyediakan data berupa URL untuk diakses oleh *client* yang dipertukarkan dalam bentuk JSON. Arsitektur REST memiliki kekurangan yaitu tidak memiliki mekanisme autentikasi yang mengakibatkan siapapun bisa mengakses, merubah, maupun menghapus data yang terdapat pada *server*. Untuk mengatasi masalah autentikasi dari arsitektur REST diperlukan sistem autentikasi. JWT merupakan token berbentuk *string* yang digunakan untuk melakukan autentikasi dan menjamin integritas pesan yang dikirim oleh salah satu pihak. Dengan menggunakan JWT pada arsitektur REST maka dapat memberikan autentikasi dan keamanan hak akses. Implementasi JWT terdapat berbagai macam algoritme *hashing* yang digunakan salah satunya algoritme HS256. Algoritme HS256 merupakan algoritme SHA256 dengan menggunakan *message authentication code* (MAC) yang disebut HMAC-SHA256. Tahun 2011 ditemukan serangan *preimage attack* dan *pseudo collission* yang memungkinkan beberapa tahun kemudian algoritme SHA256 dinyatakan tidak aman untuk digunakan. Algoritme BLAKE2S merupakan algoritme yang dibuat pada tahun 2012 dan merupakan pengembangan dari algoritme BLAKE. Algoritme BLAKE2S merupakan algoritme *hashing* yang dapat digunakan sebagai *message authentication code* (MAC). Keunggulan algoritme BLAKE2S memiliki tingkat keamanan yang lebih baik karena dibangun dengan iterasi HAIFA dan ChaCha *stream cipher*. Penelitian ini melakukan implementasi algoritme BLAKE2S pada JSON Web Token untuk mekanisme autentikasi layanan REST API sebagai alternatif dari algoritme *hash* HMAC-SHA256. Pengujian yang dilakukan berupa pengujian *test vector* dan pengujian waktu autentikasi. Pengujian *test-vector* dilakukan untuk mengetahui algoritme yang dibuat sesuai dengan ketentuan algoritme pada dokumen RFC dan didapatkan hasil yang sesuai. pengujian waktu autentikasi didapatkan algoritme BLAKE2S lebih cepat 0,981% dibanding HMAC-SHA256.

Kata kunci: Representational State Transfer, JSON Web Token, BLAKE2S, Autentikasi, Arsitektur *client-server*, Kriptografi

ABSTRACT

Bagus Satria, Implementasi Algoritme BLAKE2S Pada JSON Web Token Sebagai Algoritme *Hashing* Untuk Mekanisme Autentikasi Layanan REST API.

Pembimbing: Ari Kusyanti, S.T, M.sc dan Widhi Yahya, S.Kom, M.T

REST *is a web-based client-server communication architecture for data communications using the HTTP protocol. In* REST *architecture*, REST *server provides data in the form of* URLs *to be accessed by clients that are exchanged in* JSON *form*. REST *architecture has the disadvantage of not having an authentication mechanism that causes anyone to access, change, or delete data contained on the server*. *To solve the authentication problem from* REST *architecture is required authentication system*. JWT *is a string-based token that used to authenticate and ensure the integrity of messages sent by either party*. *By using* JWT *on the* REST *architecture then it can provide authentication and security permissions*. *Implementation of* JWT *there are various hashing algorithm used one of which algorithm* HS256. *The* HS256 *algorithm is a* SHA256 *algorithm using a message authentication code* (MAC) *called* HMAC-SHA256. *In the year* 2011 *found a preimage attack and pseudo collision attacks that allow several years later SHA256 algorithm otherwise unsafe to use*. BLAKE2S *algorithm is an algorithm created in* 2012 *and is a development of the BLAKE algorithm*. BLAKE2S *algorithm is a hashing algorithm that can be used as message authentication code* (MAC). *The advantage of the* BLAKE2S *algorithm have better security because it was built with* HAIFA *iterations and* ChaCha *stream cipher*. *This research performs* BLAKE2S *algorithm implementation on JSON Web Token for REST API service authentication mechanism as an alternative to* HMAC-SHA256 *hash algorithm*. *Testing was done by testing of test vectors and test time authentication*. *The test-vector test is performed to determine an algorithm created in accordance with the algorithm in RFC documents and obtained the corresponding results*. *the testing of authentication time obtained by BLAKE2S algorithm was 0.* 981% *faster than* HMAC-SHA256.

Keywords: *Representational State Transfer*, JSON Web Token, BLAKE2S, *Authentication*, *Client-server Architecture*, *Cryptography*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Representational State Transfer (REST) adalah standar arsitektur komunikasi *client-server* berbasis *web* untuk komunikasi data yang umumnya menggunakan protokol *Hypertext Transfer Protocol* (Fielding, 2000). REST pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000 sebagai penelitian disertasi doktoralnya. Pada arsitektur REST, REST *server* menyediakan data yang diidentifikasi oleh URL untuk diakses atau dirubah oleh REST *client* dimana data yang didapatkan atau dikirim umumnya dalam bentuk format JSON.

Arsitektur komunikasi REST memiliki beberapa kekurangan yaitu tidak ada dukungan standar untuk keamanan dan kebijakan pengaksesan data pada sisi *server*, hal ini menyebabkan siapapun bisa mengakses, merubah, maupun menghapus data yang terdapat pada sisi *server*. Untuk mengatasi kekurangan dari keamanan dan kebijakan hak akses dari arsitektur REST diperlukan sistem autentikasi untuk memberikan hak akses data pada REST *Server*. JSON Web Token atau JWT merupakan sebuah token dengan berbentuk *string* yang digunakan untuk melakukan autentikasi dan menjamin integritas pesan yang dikirim oleh salah satu pihak (Bradley,2015). Dengan menggunakan JWT pada arsitektur komunikasi REST maka dapat memberikan keamanan dan kebijakan hak akses data dari arsitektur REST.

Pada implementasi JWT terdapat berbagai macam algoritme *hashing* yang digunakan salah satunya algoritme HS256. Algoritme HS256 atau HMAC-SHA256 merupakan algoritme yang berbasis pada SHA-256 dengan menggunakan *Message Authentication Code* untuk menghasilkan sebuah *message digest*. Algoritme SHA-256 dikategorikan sebagai SHA-2 dan dirilis pada tahun 2001 digunakan untuk pengganti algoritme SHA-1 dimana pada tahun 2005 ditemukan serangan yang membuat algoritme SHA-1 dinyatakan tidak aman (Wang, 2005). Pada tahun 2011 ditemukan serangan *preimage resistance* (Khovratovich, 2011) dan *pseudo collision* (Lamberger, 2011) pada algoritme SHA-256 dan SHA-512, dimana memungkinkan beberapa tahun kemudian algoritme SHA-2 dinyatakan tidak aman untuk digunakan lagi.

Saat ini terdapat banyak algoritme *hashing* yang digunakan sebagai alternatif dari algoritme SHA-2 salah satunya adalah BLAKE2S. Algoritme BLAKE2S dirilis pada tahun 2012 yang dapat digunakan untuk algoritme *hashing* dan *message authentication code* (MAC). Algoritme BLAKE2 dibangun dengan iterasi HAIFA yang memberikan ketahanan terhadap *generic attack* seperti *herding*, *long-message second preimages* dan *length extension* (Dunkelman, 2007). Algoritme BLAKE2S memiliki tingkat keamanan yang lebih baik dibandingkan SHA-256 (Aumasson, 2013).

Hal-hal tersebut yang menjadikan dasar untuk melakukan implementasi algortime BLAKE2S pada JSON Web Token dalam mekanisme autentikasi layanan REST API. Dengan mempertimbangkan penggunaan algoritme BLAKE2S sebagai alternatif algoritme HMAC-SHA256 pada JSON Web Token diharapkan memberikan alternatif algoritme pada JSON Web Token. Mekanisme pengujian penelitian ini dengan cara pengujian *test vector* pada algoritme BLAKE2S, pengujian fungsional sistem dengan *integration testing*, pengujian waktu autentikasi untuk mengetahui perbedaan waktu antar algoritme yang digunakan, dan pengujian keamanan dengan menggunakan JWT dan tidak pada arsitektur REST.

1.2 Rumusan masalah

Berdasarkan latar belakang penelitian dapat disimpulkan rumusan masalah yaitu :

1. Bagaimana implementasi algoritme BLAKE2S pada JSON Web Token untuk sistem autentikasi berbasis token pada arsitektur REST-API?
2. Bagaimana hasil perbandingan performa waktu autentikasi penggunaan algoritme BLAKE2S pada JSON Web Token ?

1.3 Tujuan

Tujuan dari penelitian ini adalah

1. Menerapkan algoritme BLAKE2S pada pustaka JSON Web Token sebagai alternatif algoritme *hash* HMAC-SHA265.
2. Mengetahui pengaruh algoritme BLAKE2S terhadap perbedaan waktu autentikasi.

1.4 Manfaat

Diharapkan dari penelitian ini dapat diperoleh algoritme alternatif pengganti HMAC-SHA256 yang diterapkan pada JSON Web Token sehingga memberikan keamanan autentikasi dan integritas data yang lebih baik.

1.5 Batasan masalah

Batasan masalah dalam penelitian ini diberikan dengan tujuan agar pembahasan tidak melebar dan lebih terperinci. Adapun penelitian ini dibatasi oleh hal-hal berikut:

1. Implementasi algoritme BLAKE2S menggunakan bahasa pemrograman Javascript NodeJS yang diterapkan pada bagian *server-side* REST.
2. Implementasi JSON Web Token menggunakan pustaka yang disediakan oleh *Node Package Manager* (NPM).
3. Pengiriman data pada arsitektur REST dilakukan dengan menggunakan protokol HTTP.

1.6 Sistematika pembahasan

Skripsi ini dibagi menjadi 6 bab, masing masing bab akan diuraikan sebagai berikut :

BAB I PENDAHULUAN

Dalam bab ini dijelaskan tentang latar belakang pemilihan judul Implementasi Algoritme BLAKE2S pada JSON Web Token (JWT) Sebagai Algoritme *Hashing* Untuk Mekanisme Autentikasi Layanan REST-API, rumusan masalah dari pemilihan judul, tujuan penelitian, manfaat penelitian, dan sistematika pembahasan.

BAB II LANDASAN KEPUSTAKAAN

Bab ini menjelaskan tentang penelitian yang telah dilakukan mengenai Algoritme BLAKE2S maupn HMAC-SHA256 dan juga berisi dasar teori yang diperlukan untuk mendukung penelitian ini.

BAB III METODOLOGI

Dalam bab ini dijelaskan tentang langkah-langkah apa saja yang akan digunakan dalam melakukan implementasi JSON Web Token dengan algoritme BLAKE2S pada arsitektur REST.

BAB IV PERANCANGAN

Dalam bab ini dijelaskan tentang rancangan algoritme BLAKE2S, cara kerja JSON Web Token dan arsitektur *client-server* REST dengan menggunakan JSON Web Token.

BAB V IMPLEMENTASI

Dalam bab ini dilakukan implementasi algoritme BLAKE2S dengan menggunakan bahasa pemrograman Javascript NodeJS dan implementasi REST *server* dengan *library* Express. Kemudian dilakukan integrasi algoritme BLAKE2S dengan JSON Web Token yang digunakan pada arsitektur *client server* REST.

BAB VI PENGUJIAN DAN ANALISIS

Dalam bab ini dijelaskan tentang pengujian dari sistem yang sudah dibuat. Pengujian dilakukan dengan uji test vector, pengujian fungsional sistem, dan pengujian waktu autentikasi.

BAB VII PENUTUP

Dalam bab ini memuat kesimpulan dari penerapan algoritme BLAKE2S pada JSON Web Token untuk arsitektur *client-server* REST yang telah diperoleh dari penelitian ini dan juga saran yang dapat digunakan untuk pengembanga selanjutnya.

BAB 2 KAJIAN PUSTAKA DAN DASAR TEORI

Kajian Pustaka dan Dasar teori merupakan bentuk kerangka teori yang berdasarkan literatur yang berhubungan dengan masalah yang diteliti. Bab ini membahas tinjauan pustaka yang digunakan untuk penelitian skripsi, dimana dalam tinjauan pustaka terdapat kajian pustaka mengenai referensi dari penelitian yang sudah dilakukan sebelumnya. Sedangkan dasar teori membahas mengenai teori-teori yang berhubungan dengan penelitian yang akan dilakukan.

2.1 Kajian Pustaka

Pada sub bab ini membahas penelitian yang akan dilakukan menggunakan algoritme BLAKE2S sebagai fungsi *hash*. Pada penelitian yang berjudul "*BLAKE2: simpler, smaller, fast as MD5*" (Aumasson, 2013) algoritme BLAKE2 dibangun dengan menggunakan iterasi HAIFA yang memberikan ketahanan terhadap *generic attack* seperti *herding*, *long-message second preimages* dan *length extension* (Dunkelman, 2007). Selain algoritme BLAKE (pendahulu algoritme BLAKE2) memiliki predikat "*very large security margin*" yang dilakukan evaluasi oleh *National Institute of Standards and Technology* dalam kompetisi SHA-3 (NIST,2015), dimana memiliki tingkat keamanan algoritme yang baik berdasarkan kriteria fungsi *hash* yaitu tahan terhadap *collision*, *preimage attack* dan *second preimage attack*.

Dalam komunikasi aristektur REST diperlukan JSON Web Token dimana memberikan nilai integritas dan autentikasi terhadap pesan yang dikirim oleh salah satu pihak. JSON Web Token menggunakan algoritme HMAC-SHA256 untuk menghasilkan sebuah *signature* token tetapi telah ditemukan serangan *preimage resistance* (Khovratovich, 2011) dan *pseudo collision* (Lamberger, 2011) pada SHA256 yang memungkinkan integritas suatu data dapat dipalsukan. Pada penelitian ini penulis memutuskan untuk mengganti algoritme HMAC-SHA256 dengan algoritme BLAKE2S sebagai alternatif algoritme *hashing* pada JSON Web Token. Hal ini akan memberikan nilai integritas yang lebih baik pada *signature* JSON Web Token sehingga token tidak dapat dipalsukan dengan memanfaatkan serangan *collision* pada algoritme hash yang digunakan.

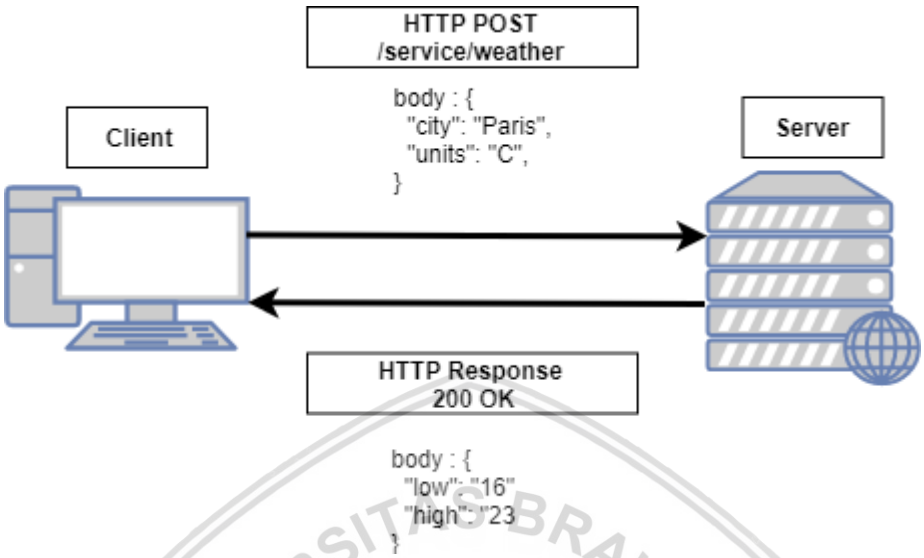
2.2 Dasar Teori

Dasar teori membahas tentang materi-materi yang berkaitan dengan penelitian implementasi algoritme BLAKE2S pada JSON Web Token untuk layanan autentikasi REST API. Dasar teori yang digunakan *Representational State Transfer* (REST), protokol HTTP, JavaScript Object Notation (JSON), JSON Web Token (JWT), *Hash Function*, dan BLAKE2S.

2.2.1 Representational State Transfer

Representational State Transfer (REST) adalah bagian dari 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 bergantung pada keadaan percakapan (World Wide Web Consortium, 2004).

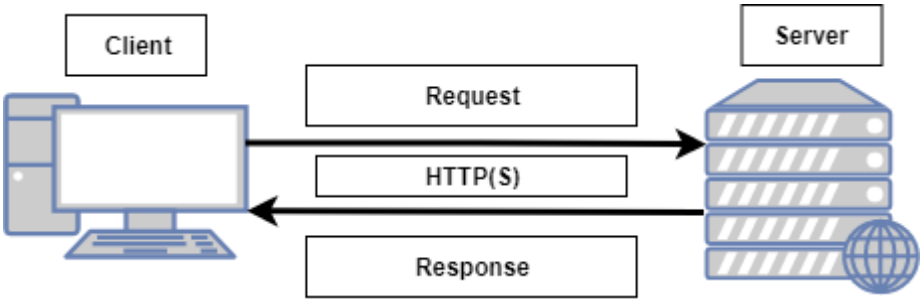


Gambar 2.1 Arsitekur REST *Client-Server*.

Arsitektur REST digunakan untuk memanipulasi data pada sebuah sistem dengan menggunakan metode protokol HTTP seperti *GET*, *POST*, *PUT*, *DELETE* dan lain sebagainya. Data diidentifikasi dengan *Uniform Resource Locator* (URL) untuk digunakan sebagai antar muka dalam memanipulasi sumber daya. Dalam arsitektur REST kembalian sumber daya dapat berupa format XML, HTML, JSON ataupun format yang lain. Dengan mengunakan protokol HTTP/HTTPS yang bersifat *stateless*, arsitektur REST ditujukan untuk *performance*, *reliability* dan *scalability*.

2.2.2 Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) merupakan prokotol aplikasi yang umumnya digunakan pada *World Wide Web* (WWW) dan merupakan fondasi dari komunikasi data WWW (Fielding, 1999). Protokol HTTP merupakan aturan komunikasi yang harus dipenuhi pada saat menggunakan protokol HTTP. Penerapan protokol HTTP diterapkan pada arsitektur *client-server* yang menggunakan pola Representational State Transfer (REST). Dalam *layer* OSI protokol HTTP terdapat pada *layer application* dan terdapat 2 jenis operasi pada protokol HTTP yaitu HTTP *request* dan HTTP *response* yang dapat diidentifikasi melalui *header* HTTP tersebut.



Gambar 2.2 Arsitektur Komunikasi *Client-Server* melalui protokol HTTP.

Hypertext Transfer Protocol Secure (HTTPS) merupakan ekstensi protokol HTTP dimana menggunakan protokol *Transport Layer Security* (TLS) untuk mengenkripsi data komunikasi.

2.2.3 Javascript Object Notation (JSON)

Javascript Object Notation (JSON) merupakan format pertukaran data antar mesin yang mudah untuk dibaca dan ditulis oleh manusia (JSON.org). JSON dibuat berdasarkan bagian dari pemrograman Javascript dan di standarkan pada dokumen ECMA-263.

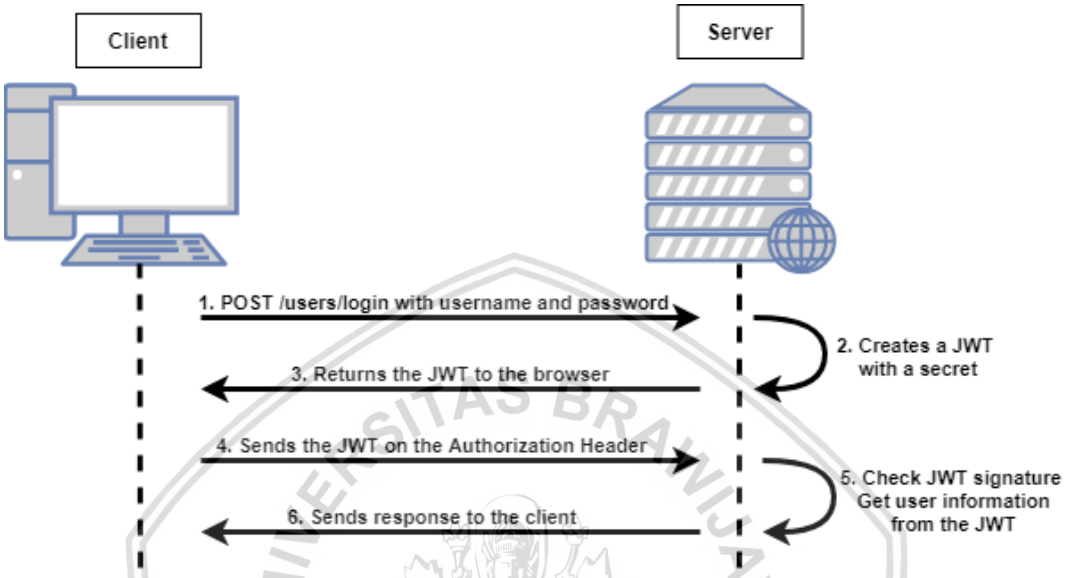
```
{
  "array": [
    1,
    2,
    3
  ],
  "boolean": true,
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d",
    "e": "f"
  },
  "string": "Hello World"
}
```

Gambar 2.3 Contoh Format JavaScript Object Notation.

JSON merupakan format yang tidak bergantung dengan bahasa pemrograman apapaun karena menggunakan gaya bahasa yang umum, maka dari itu JSON ideal sebagai format pertukaran data. Struktur JSON dinyatakan sebagai objek dimana anggota objek didalamnya berupa pasangan *key* dan *value* ataupun dapat berupa objek JSON dimana dipisahkan dengan tanda koma.

2.2.4 JSON Web Token (JWT)

JSON Web Token (JWT) merupakan *access* token dengan *format JavaScript Object Notation* (Bradley, 2015). Token tersebut digunakan sebagai autentikasi token pada *client-server*,token tersebut didesain ringkas dan *url-safe*.



Gambar 2.4 Mekanisme Autentikasi dengan menggunakan JWT.

JSON Web Token tersusun atas 3 bagian yaitu *header*, *payload* dan *signature*. Bagian *header* berisi informasi tentang *signature* JWT untuk diproses validasi JWT. Terdapat 2 pasang nilai yaitu typ dan alg. Nilai dari typ digunakan sebagai penanda bahwa data tersebut merupakan JWT dan nilai dari alg merupakan algoritme yang digunakan untuk membuat *signature* atau enkripsi JWT. Bentuk akhir dari *payload* dirubah dalam *encoding* base64Url. Algoritme yang dapat digunakan untuk membuat *signature* dalam JWT dibahas dalam JWS dan untuk membuat enkripsi dibahas dalam JWE.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

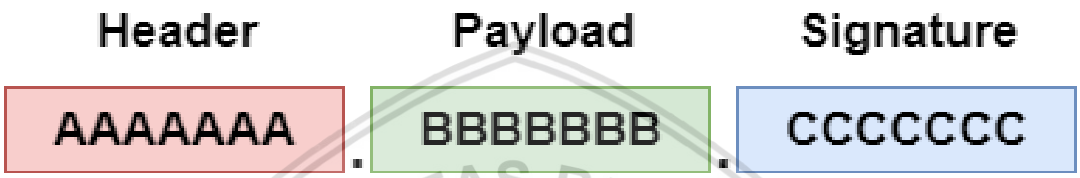
Gambar 2.5 *Header* JSON Web Token.

Payload pada JWT merupakan komponen dimana data dimasukan kedalam *payload*. Data dalam *payload* merupakan data aplikasi yang disematkan dalam *payload* sesuai kebutuhan aplikasi, data tersebut harus berupa pasangan *key* dan *value*. *Payload* kemudian dilakukan *encoding* ke dalam bentuk base64url. Panjang dari *payload* bergantung pada panjang data yang dimasukkan pada *payload* itu sendiri.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Gambar 2.6 *Payload* JSON Web Token.

Pada *Signature* JWT digunakan untuk membuat nilai *digest* dari *header* dan *payload*. Panjang dari signature sejumlah 43 karakter apabila menggunakan fungsi *hash* yang memiliki bit sebesar 256.



Gambar 2.7 Susunan JSON Web Token.

2.2.4.1 *JSON Web Signature (JWS)*

JSON Web Signature (JWS) merupakan bagian penting dalam JWT. JWS mengatur algoritme *signature* atau *message authentication code* (MAC) yang didukung oleh JWA dan melakukan autentikasi pesan JWT untuk menentukan pesan tersebut tidak diubah dalam media pengiriman. Hal ini memberikan integritas data yang dikirim.

JWS merupakan format yang dapat diseragamkan dan dapat dibaca oleh mesin. JWS dapat digunakan untuk mengirim informasi dari satu situs web ke situs lain, dan terutama ditujukan untuk komunikasi di web.

2.2.4.2 *JSON Web Encryption (JWE)*

JSON Web Encryption (JWE) adalah standar IETF yang menyediakan sintaks standar untuk pertukaran data terenkripsi, berdasarkan JSON dan Base64. JWE digunakan untuk menjaga pesan JWT agar tetap rahasia, tidak bisa dibaca atau dipahami oleh oleh pihak yang tidak memiliki kunci.

JWE bertindak dengan melakukan enkripsi pada JWT dan melakukan validasi JWT dengan melakukan dekripsi pesan JWT berdasarkan algoritme yang disediakan oleh JWA.

2.2.4.3 *JSON Web Algorithm (JWA)*

JSON Web Algorithm (JWA) merupakan kumpulan spesifikasi algoritme yang dapat dipakai pada JSON Web Signature (JWS) dan JSON Web Encryption (JWE). Beberapa algoritme yang sudah diregisterasi diantaranya HMAC-SHA256, RSA dan lain sebagainya.

Pada JWA terdapat algoritme enkripsi dan juga algoritme *hashing* dimana algoritme enkripsi digunakan untuk merahasiakan data dan algoritme *hashing* digunakan untuk memastikan integritas data.

2.2.5 Hash Function

Fungsi *hash* adalah fungsi yang menerima masukan string yang panjangnya sembarang dan mengkonversinya menjadi string keluaran yang panjangnya tetap (Munir, 2004). Hasil yang dikeluarkan dari fungsi hash disebut *hash value*, *hash codes*, *digest* atau *hashes* tetapi pada umumnya disebut *message digest*. Fungsi *hash* bersifat *one way function* yang artinya hasil hash tidak bisa dikembalikan ke bentuk semula.

Komponen utama fungsi *hash* ialah *hash table* yang merupakan struktur data yang digunakan untuk memetakan sebuah nilai menjadi *hashes*. Fungsi *hash* sangat berguna dalam bidang kriptografi seperti pencocokan data dengan hasil *digest* nya. Fungsi *hash* memberikan nilai integritas data oleh karena itu banyak diterapkan pada bidang kriptografi diantaranya *checksum*, *digital signature* dan lain sebagainya. Dengan membandingkan hasil perhitungan *hash* dengan nilai *hash* yang telah diberikan seseorang dapat mengetahui apakah data tersebut telah dimodifikasi atau tidak.

2.2.6 Hash-Based Message Authentication Code (HMAC)

Hash-Based Message Authentication Code (HMAC) merupakan sebuah tipe dari *Message Authentication Code* (MAC) yang dipadukan dengan sebuah fungsi *hash* dimana memungkinkan masukan berupa data dan kata kunci rahasia untuk menghasilkan sebuah *digest* (Bellare, 1996). HMAC memberikan nilai integritas data dan autentikasi dalam kriptografi tetapi HMAC tidak memberikan nilai kerahasiaan karena HMAC tidak mengenkripsi pesan.

$$\mathrm{HMAC(K,M)} = \mathrm{H((K' \oplus opad)||(H((K' \oplus ipad)||M))} \quad (2.1)$$

Dimana :

H = Fungsi *Hash*.

K = Kunci Rahasia.

M = Pesan yang akan di *hash*.

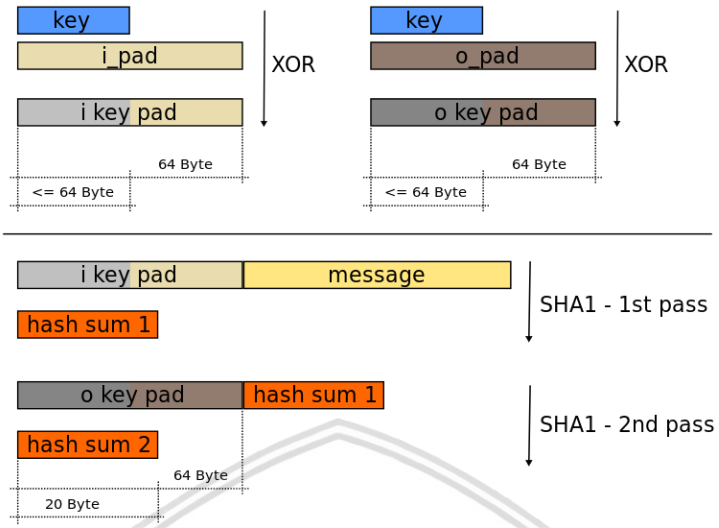
K’ = Turunan dari K dimana ditambahkan angka 0 ke arah kanan pada blok inputan dari fungsi *hash* atau hasil *hash* dari K apabila K lebih panjang dari ukuran blok dari fungsi *hash*.

|| = Penggabungan / penyambungan

$\oplus$ = Exclusive Or

Opad = *Outer Padding* (0x5c5c..5c, panjang satu blok konstan heksadesimal)

Ipad = *Inner Padding* (0x3636..36, panjang satu blok konstan heksadesimal)



Gambar 2.8 Algoritme HMAC

Sumber: https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

2.2.7 *Secure Hash Algorithm*

Secure Hash Algorithm atau yang dikenal SHA merupakan sebuah fungsi kriptografi *hash* yang dibuat oleh *National Institute of Standard and Technology* (Munir, 2004). Algoritme SHA-2 merupakan algoritme pengembangan dari algoritme SHA-1 dimana ukuran hasil keluaran SHA-2 lebih panjang dibanding SHA-1. Jenis SHA-2 dibagi menjadi 4 berdasarkan hasil keluaran yaitu 224, 256, 384 dan 512. Pembagian tersebut dengan tujuan membagi berdasarkan tingkat keamanan untuk data yang akan di *hash*.

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Gambar 2.9 Tabel Jenis Secure Hash Algorithm

Sumber : http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=910977

Setiap algoritme dibagi menjadi 2 tahap yaitu *preprocessing* dan *hash computation*. Tahap *preprocessing* dimulai dengan memberikan *padding* pada pesan, membagi pesan yang sudah diberi *padding* kedalam m-bit blok dan inisialisasi nilai yang akan digunakan saat *hash computation*. *Hash computation* membuat *message schedule* dari pesan yang telah diberi *padding* dan menggunakan *message schedule* tersebut bersama dengan fungsi *hash*, konstan dan operasi kalimat secara iteratif untuk menghasilkan *hash value* atau *digest*.

2.2.8 SHA-256

Algoritme SHA-256 digunakan untuk membuat *message digest* dimana panjang *message* harus diantara $0 \leq \ell \leq 2^{64}$. Algoritme ini menggunakan sebuah *message schedule* yang terdiri dari 64 elemen 32-bit word yang diberi label *W0, W1 .. W63*. Delapan buah variabel 32-bit yang diberi label *a, b, c, d, e, f ,g ,h*. Variabel penyimpan nilai hasil *hash* berjumlah 8 buah *word* 32-bit yang diberi label $H_0^i,\ H_1^i\ ..H_7^i$.

Algoritme SHA-256 mengubah masukan ke dalam *message digest* 256 bit. Berdasarkan *Secure Hash Signature Standard*, pesan masukan yang panjangnya lebih pendek dari 2^{64}, harus dioperasikan oleh 512 bit dalam kelompok dan menjadi sebuah *message digest* 256-bit.

Tahapan algoritme SHA-256 :

1.*Message Padding*: pesan berupa biner disisipkan dengan angka 1 dan ditambahkan bit 0 hingga panjang pesan kongruen dengan 448 modulo 512. Panjang pesan yang asli kemudian ditambahkan sebagai angka biner 64 bit maka panjang pesan sekarang menjadi kelipatan 512 bit.

2.*Parsing*: Pesan yang sudah di *padding* tadi kemudian dibagi menjadi N buah blok dimana tiap blok memiliki panjang 512 bit.

3.*Message Expansion*: Masing-masing 512 bit blok lalu dipecah menjadi 16 *word* 32 bit ($M_0^i,\ M_1^i\ ..M_{15}^i$) yang mana nantinya diperluas menjadi 64 *word* yang diberi label *W0, W1 .. W63.*

4. *Message compression*: Masing-masing dari 64 bit diproses dengan algoritme *hash* SHA-256. Dalam proses algoritme SHA-256 , fungsi hash membuat 8 variabel yang diberi label $H_0^i,\ H_1^i\ ..H_7^i$ dan diberikan nilai awal seperti pada Tabel 2.1.

Tabel 2.1 Nilai Variabel H

$A = H_0^{(0)}$	6a09e667
$\mathrm{B} = H_1^{(0)}$	bb67ae85
$C = H_2^{(0)}$	3c6ef372
$D = H_3^{(0)}$	a54ff53a
$E = H_4^{(0)}$	510e527f

Tabel 2.2 Lanjutan Nilai Variabel H

$F = H_5^{(0)}$	9b05688c
$G = H_6^{(0)}$	1f83d9ab
$H = H_7^{(0)}$	5be0cd19

5. Algoritme ini melakukan perhitungan sebanyak 64 kali putaran untuk setiap perhitungan blok. Delapan variabel yang diberi label A, B, C .., H nilainya akan terus berganti selama perputaran sebanyak 64 kali.

$$T_1 = H + \Sigma_1(E) + Ch(E,F,G)[1] + Kt + Wt \quad (2.2)$$

$$T_2 = \Sigma_0(A) + Maj(A,B,C)[1] \quad (2.3)$$

$$H = G \quad (2.4)$$

$$G = F \quad (2.5)$$

$$F = E \quad (2.6)$$

$$E = D + T_1 \quad (2.7)$$

$$D = C \quad (2.8)$$

$$C = B \quad (2.9)$$

$$B = A \quad (2.10)$$

$$A = T_1 + T_2 \quad (2.11)$$

6. Setelah perputaran sebanyak 64 kali, nilai *hash* $H^{(i)}$ kemudian dihitung sebagai berikut :

$$H_0^{(i)} = a + H_0^{(i-1)} \quad (2.12)$$

$$H_1^{(i)} = b + H_1^{(i-1)} \quad (2.13)$$

$$H_2^{(i)} = c + H_2^{(i-1)} \quad (2.14)$$

$$H_3^{(i)} = d + H_3^{(i-1)} \quad (2.15)$$

$$H_4^{(i)} = e + H_4^{(i-1)} \quad (2.16)$$

$$H_5^{(i)} = f + H_5^{(i-1)} \quad (2.17)$$

$$H_6^{(i)} = g + H_6^{(i-1)} \quad (2.18)$$

$$H_7^{(i)} = h + H_7^{(i-1)} \quad (2.19)$$

7. Selanjutnya hasil akhir SHA-256 didapat dari penggabungan delapan variabel yang sudah dikomputasi.

$$H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)} || H_5^{(N)} || H_6^{(N)} || H_7^{(N)} \quad (2.20)$$

2.2.9 BLAKE2

Algoritme BLAKE2 merupakan fungsi *hash* dimana algoritme tersebut pengembangan dari algoritme BLAKE. BLAKE2 dipublikasikan pada 21 desember 2012 dan dibuat oleh Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, dan Christian Winnerlein. Algoritme BLAKE2 dibangun dengan menggunakan Dan Bernstein's ChaCha *stream cipher* dan sedikit perubahan dimana salinan permutasi dari blok input akan dioperasikan dengan exclusive or (XOR) bersama *round constant*, dimana hasil tersebut akan disematkan sebelum operasi ChaCha (Aumasson, 2013). Operasi ChaCha berlangsung pada *array word* ukuran 4 x 4.

Algoritme BLAKE2 dapat digunakan untuk *keying hash* atau MAC, *salting*, *personalization*, dan *hash tree mode*. Dalam implementasinya algoritme BLAKE2 dibagi menjadi BLAKE2B dimana *output* yang dihasilkan hingga 64 *bytes* dan BLAKE2S yang menghasilkan *output* hingga 32 *bytes*.

2.2.10 BLAKE2S

Algoritme BLAKE2S didesain untuk berjalan pada perangkat 32 bit yang dapat menghasilkan *message digest* dengan panjang hingga 32 *bytes*. Algoritme BLAKE2S dapat menerima input sebesar ℓ, dimana $0 \leq \ell \leq 2^{64}$.

Sebelum data masukan di proses dilakukan *padding* terlebih dahulu. *Padding* dilakukan pada blok terakhir dari sebuah data apabila diperlukan dengan *null byte*. Apabila panjang data kelipatan dari panjang blok, maka tidak perlu dilakukan *padding*. Tujuan dilakukan *padding* untuk membuat serangkaian urutan N dimana nilainya $N = \lceil \ell/128 \rceil$ 16-*word* blok dengan notasi $m^0 m^1 .. m^{N-1}$ dan dilakukan *hash* dengan langkah:

$h^0 \leftarrow IV \oplus P$

$\boldsymbol{for}\ i = 0, \ldots, N-1$

$\quad h^{i+1} \leftarrow \boldsymbol{compress}\ (h^i, m^i, \ell^i)$

$\boldsymbol{return}\ h^N$ (2.21)

Notasi ℓ^i merupakan representasi angka dari data pada $m^0, m^1, .., m^i$. Notasi P merupakan parameter blok berupa panjang *digest* dengan nilai minimal 1 *byte* dan maksimal 32 *byte*, panjang *key* hingga 32 *byte* (opsional). *Initialization Vector* merupakan nilai konstan yang nilainya diberikan secara statis yang diberikan pada Tabel 2.2.

Tabel 2.3 Nilai Initialization Vector BLAKE2S.

$IV_0 = 6a09e667$	$IV_4 = 510e527f$
$IV_1 = bb67ae85$	$IV_5 = 9b05688c$
$IV_2 = 3c6ef372$	$IV_6 = 1f83d9ab$
$IV_3 = a54ff53a$	$IV_7 = 5be0cd19$

Pada fungsi *compress* menerima parameter masukan berupa h^i, m^i, ℓ^i, *counter* dengan label $t = t_1, t_2$ dan sebuah *flag* dengan label f_0, f_1 untuk melakukan komputasi

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ IV_0 & IV_1 & IV_2 & IV_3 \\ t_0 \oplus IV_4 & t_1 \oplus IV_5 & t_2 \oplus IV_6 & t_3 \oplus IV_7 \end{pmatrix} \quad (2.22)$$

Keadaan internal dari v akan berubah secara berurutan selama 10 *round*, dimana setiap *round* akan melakukan fungsi G seperti pada Tabel 2.3

Tabel 2.4 Fungi G dalam BLAKE2S.

$G_0(v_0, v_4, v_8, v_{12})$	$G_4(v_0, v_5, v_{10}, v_{15})$
$G_1(v_1, v_5, v_9, v_{13})$	$G_5(v_1, v_6, v_{11}, v_{12})$
$G_2(v_2, v_6, v_{10}, v_{14})$	$G_6(v_2, v_7, v_8, v_{13})$
$G_3(v_3, v_7, v_{11}, v_{15})$	$G_7(v_3, v_4 8, v_9, v_{14})$

Dimana algoritme fungsi G sebagai berikut:

$$\begin{aligned} a &\leftarrow a + b + m_{\sigma\tau(2i)} \\ d &\leftarrow (d \oplus a) \ggg 16 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg 12 \\ a &\leftarrow a + b + m_{\sigma\tau(2i+1)} \\ d &\leftarrow (d \oplus a) \ggg 8 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg 7 \end{aligned} \quad (2.23)$$

Pada fungsi G dibutuhkan nilai konstan permutasi pada Tabel 2.4.

Tabel 2.5 Nilai *Round Constant* BLAKE2S.

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Setelah 10 *round* selesai dilakukan maka hasil *digest* didapat dengan cara menggabungkan nilai $h'_0, \dots, h'_7$ secara berurutan.

$$h'_0 \leftarrow h_0 \oplus v_0 \oplus v_8 \quad (2.24)$$

$$h'_1 \leftarrow h_1 \oplus v_1 \oplus v_9 \quad (2.25)$$

$$h'_2 \leftarrow h_2 \oplus v_2 \oplus v_{10} \quad (2.26)$$

$$h'_3 \leftarrow h_3 \oplus v_3 \oplus v_{11} \quad (2.27)$$

$$h'_4 \leftarrow h_4 \oplus v_4 \oplus v_{12} \quad (2.28)$$

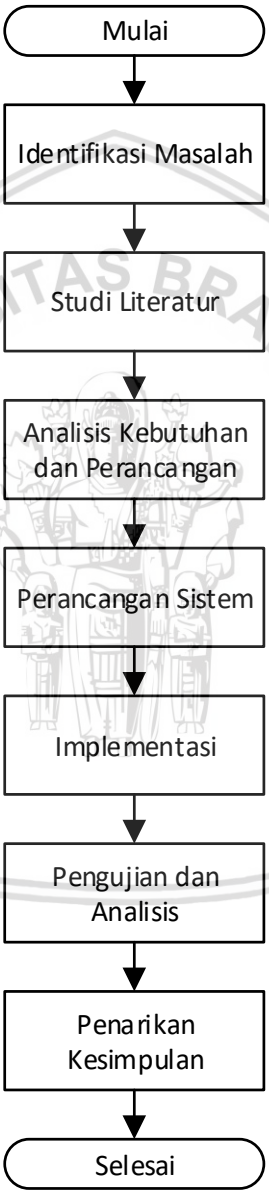
$$h'_5 \leftarrow h_5 \oplus v_5 \oplus v_{13} \quad (2.29)$$

$$h'_6 \leftarrow h_6 \oplus v_6 \oplus v_{14} \quad (2.30)$$

$$h'_7 \leftarrow h_7 \oplus v_7 \oplus v_{15} \quad (2.31)$$

BAB 3 METODOLOGI

Bab ini menjelaskan metode yang digunakan dalam penelitian yang dilakukan, yaitu identifikasi masalah, studi literatur, analisis kebutuhan dan perancangan sistem, implementasi, pengujian dan analisis, dan penarikan kesimpulan. Langkah-langkah penelitian digambarkan pada Gambar 3.1 diagram alir.



Gambar 3.1 Diagram Alir Penelitian.

3.1 Identifikasi Masalah

Hal pertama yang perlu dilakukan yaitu melakukan identifikasi masalah. Masalah yang telah dijelaskan pada latar belakang yaitu bagaimana implementasi algoritme BLAKE2S dengan bahasa pemrograman Javascript pada JSON Web Token sebagai alternatif algoritme HMAC-SHA256 untuk autentikasi layanan REST-API. Kemudian bagaimana perbandingan waktu autentikasi dengan menggunakan algoritme BLAKE2S dengan algoritme HMAC-SHA256. Setelah menentukan permasalahan yang ingin diteliti, kemudian dilanjutkan dengan studi literatur.

3.2 Studi Literatur

Untuk mendalami permasalahan yang diteliti, maka peneliti membutuhkan informasi-informasi terkait permasalahan yang telah dijelaskan pada latar belakarng penelitian. Informasi-informasi tersebut dapat diperoleh melalui buku, jurnal, situs internet. Adapun literatur studi yang dilakukan selama penelitian meliputi :

1. Membaca dan memahami alur kerja algoritme BLAKE2S berdasarkan dokumen RFC 7693 dan jurnal "*BLAKE2: simpler, smaller, fast as MD5*".
2. Membaca dan memahami komponen penyusun JSON Web Token yang bersumber dari dokumen RFC 7519.
3. Membaca dan memahami konsep dan teori dari Representational State transfer atau REST API.

3.3 Analisis Kebutuhan

Analisis kebutuhan merupakan kegiatan untuk mengetahui sebuah kebutuhan yang harus dipenuhi dalam penelitian ini. Analisis kebutuhan penelitian ini dilakukan dengan mengidentifikasi kebutuhan dari sistem aplikasi *web* REST API *client-server* dengan menggunakan JWT dan algoritme BLAKE2S sebagai fungsi *hash*.

Selain menganalisis kebutuhan dari sistem REST API dilakukan juga analisis kebutuhan yang dibutuhkan oleh algoritme BLAKE2S agar dapat digunakan pada JSON Web Token. Pada algoritme BLAKE2S dilakukan analisis kebutuhan terkait nilai variabel *input* yang dibutuhkan oleh algoritme BLAKE2S dan jenis *output* yang dihasilkan.

3.4 Perancangan Sistem

Perancangan sistem dilakukan apabila tahap analisis kebutuhan telah terselesaikan. Langkah dalam perancangan sistem yaitu perancangan diagram alir sistem aplikasi REST API *client-server*, bahasa pemrograman yang digunakan, *library* yang digunakan. Penelitian bersifat implementatif perancangan yang artinya sistem yang dirancang akan diimplementasikan.

Perancangan diagram alir sistem dilakukan untuk menggambarkan proses autentikasi *client* pada *server* REST API melalui protokol HTTP dan mengembalikan sebuah token dengan dilakukan *hash* pada token tersebut.

3.5 Implementasi

Tahap implementasi merupakan tindak lanjut dari perancangan yang sudah di susun sebelumnya. Proses implementasi dilakukan untuk membuat model sistem yang telah dirancang pada perancangan sistem menjadi sebuah aplikasi yang dapat memenuhi kebutuhan dan menyelesaikan permasalahan pada latar belakang. Proses implementasi dimulai dengan menjabarkan spesifikasi aplikasi mulai dari arsitektur REST, bahasa pemrograman yang digunakan, spesifikasi algoritme yang digunakan, dan *library* atau perangkat lunak pendukung yang digunakan pada sistem.

3.6 Pengujian dan Analisis

Pada tahap ini, sistem berhasil diimplementasikan. Pengujian dilakukan untuk menguji perangkat lunak apakah dapat berjalan sesuai dengan tujuan dan kebutuhan yang telah dirancang sebelumnya. Strategi pengujian perangkat lunak yang telah dirancang antara lain :

1. Pengujian *test vector* digunakan untuk menguji hasil *message digest* fungsi *hash* BLAKE2S yang dihasilkan agar sesuai dengan hasil *output* fungsi *hash* pada dokumen RFC 7693.
2. Pengujian fungsional sistem digunakan untuk menguji apakah kebutuhan fungsional benar sesuai tujuan dan menghasilkan hasil sesuai tujuan dari kebutuhan fungsional.
3. Pengujian autentikasi digunakan untuk mengetahui perbedaan dari algoritme yang digunakan dalam menghasilkan token yang digunakan autentikasi.

3.7 Pengambilan Kesimpulan dan Saran

Pengambilan kesimpulan dan saran merupakan tahap terakhir pada penelitian yang dilakukan. Kesimpulan dari hasil pengujian terhadap aplikasi yang telah dibangun. Saran dimaksudkan untuk memperbaiki kekurangan yang terjadi dan menyempurnakan penulisan serta pengembangan penelitian yang lebih lanjut.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

Bab ini bertujuan untuk menjelaskan analisis kebutuhan sistem dan rancangan pengujian yang dilakukan pada proses pengujian. Adapun rancangan yang dijabarkan adalah alur kerja sistem dan perancangan penujian.

4.1 Analisis Kebutuhan

Analisis kebutuhan ialah penguraian dari suatu pembahasan yaitu analisis mengenai perancangan sistem autentikasi REST API dengan JSON Web Token yang menggunakan algoritme BLAKE2S sebagai algoritme *hash* pada token JSON Web Token. Dalam penelitian ini dilakukan perancangan algoritme BLAKE2S dan perancangan arsitektur REST *client-server*.

4.1.1 Kebutuhan Fungsional

Kebutuhan fungsional sistem menjelaskan hal-hal yang harus disediakan oleh sebuah sistem. Kebutuhan fungsional REST dari implementasi algoritme BLAKE2S pada JSON Web Token sebagai algoritme *hashing* untuk mekanisme autentikasi layanan REST-API sebagai berikut

1. *Client* dapat melakukan autentikasi (*login*/*register*) pada *server* REST dengan menggunakan *email* dan *password*.
2. *Server* REST dapat memberikan token JWT apabila autentikasi berhasil dilakukan.
3. *Server* REST dapat melakukan validasi token JWT yang diterima dari *client*.
4. *Client* dapat mengakses data informasinya dengan menggunakan token yang diberikan oleh server REST.

4.1.2 Kebutuhan Perangkat Keras

1. *Server* REST
    a. Perangkat Laptop.
2. *Client*
    a. Perangkat Laptop.
3. *Database* MongoDB
    a. Perangkat Laptop.

4.1.3 Kebutuhan Perangkat Lunak

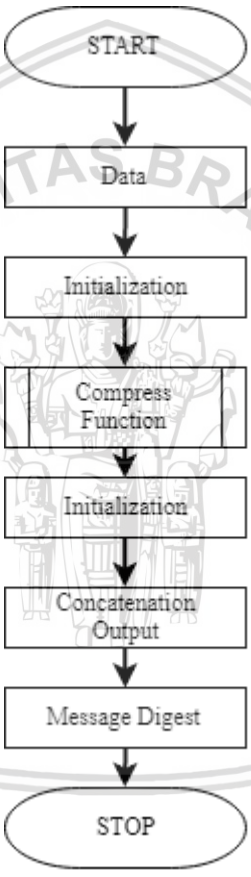
1. *Server* REST
    a. Express JS.
    b. JSON Web Token.

2. *Client*
    a. Postman.

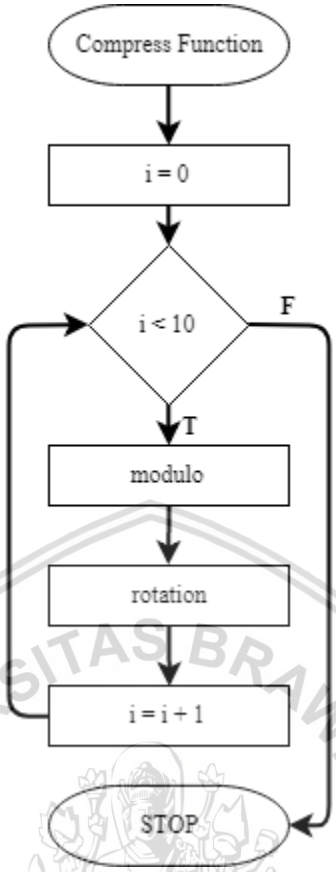
4.2 Perancangan Sistem

4.2.1 Perancangan Algoritme

Fungsi dari perancangan algortime digunakan untuk mempermudah dalam proses implementasi algoritme dan juga sebagai gambaran alur kerja algoritme BLAKE2S yang dibuat sesuai dengan algoritme asli dari BLAKE2S. Berikut adalah sekma global dari algoritme BLAKE2S.



Gambar 4.1 Skema Global Algoritme BLAKE2S.



Gambar 4.2 Skema G *Function.*

Penjelasan skema global dari algoritme BLAKE2S pada Gambar 4.1 dan Gambar 4.2 sebagai berikut

1. Data merupakan masukan untuk algoritme BLAKE2S dengan maksimum panjang pesan 2^{64} dan kunci rahasia dengan panjang maksimum 32 karakter.
2. Inisialisasi dilakukan untuk mengisi *word state* ukuran 16 x 16 dengan ketentuan yang sudah diatur pada algoritme tersebut.
3. Fungsi *compress* menerima masukan berupa data yang diolah pada 10 putaran dimana setiap putaran melakukan fungsi G sebanyak 8 kali. Hasil dari *compress function* digunakan untuk menghasilkan *message digest.*
4. *Concatenation* dilakukan untuk menggabungkan hasil dari *compress function* dengan nilai inisialisasi *chain value* untuk menghasilkan nilai *message digest*.

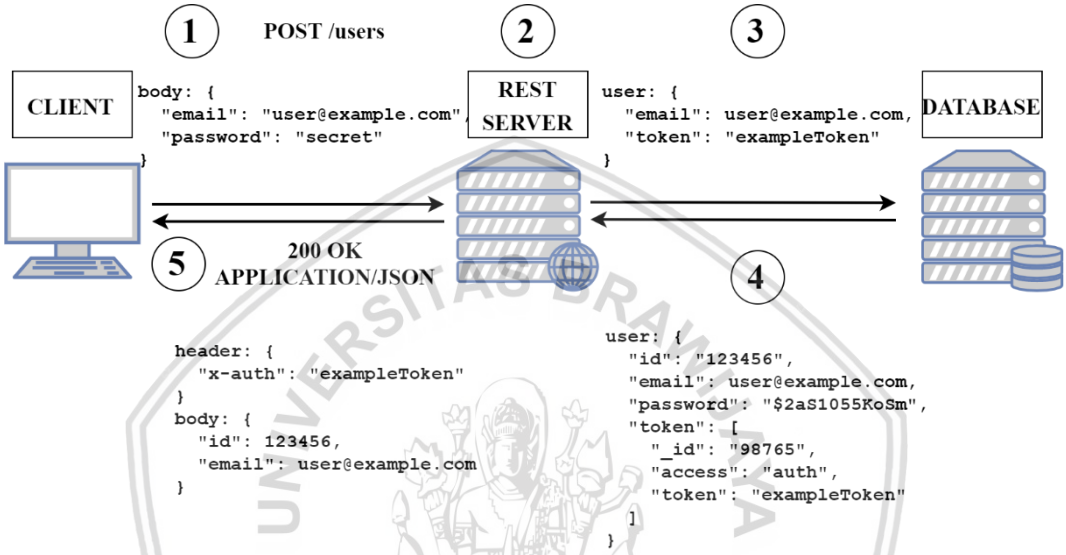
Hasil manualisasi dengan input “abc” tanpa kunci pada algoritme BLAKE2S yang diperoleh ditampilkan pada Tabel 4.1.

Tabel 4.1 Hasil manualisasi input algoritme BLAKE2S.

m[16]	00636261 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
(i=0)v[16]	6B08E647 BB67AE85 3C6EF372 A54FF53A 510E527F 9B05688C 1F83D9AB 5BE0CD19 6A09E667 BB67AE85 3C6EF372 A54FF53A 510E527C 9B05688C E07C2654 5BE0CD19
(i=1) v[16]	16A3242E D7B5E238 CE8CE24B 927AEDE1 A7B430D9 93A4A14E A44E7C31 41D4759B 95BF33D3 9A99C181 608A3A6B B666383E 7A8DD50F BE378ED7 353D1EE6 3BB44C6B
(i=2) v[16]	3AE30FE3 0982A96B E88185B4 3E339B16 F24338CD 0E66D326 E005ED0C D591A277 180B1F3A FCF43914 30DB62D6 4847831C 7F00C58E FB847886 C544E836 524AB0E2
(i=3) v[16]	7A3BE783 997546C1 D45246DF EDB5F821 7F98A742 10E864E2 D4AB70D0 C63CB1AB 6038DA9E 414594B0 F2C218B5 8DA0DCB7 D7CD7AF5 AB4909DF 85031A52 C4EDFC98
(i=4) v[16]	2A8B8CB7 1ACA82B2 14045D7F CC7258ED 383CF67C E090E7F9 3025D276 57D04DE4 994BACF0 F0982759 F17EE300 D48FC2D5 DC854C10 523898A9 C03A0F89 47D6CD88
(i=5) v[16]	C4AA2DDB 111343A3 D54A700A 574A00A9 857D5A48 B1E11989 6F5C52DF DD2C53A3 678E5F8E 9718D4E9 622CB684 92976076 0E41A517 359DC2BE 87A87DDD 643F9CEC
(i=6) v[16]	3453921C D7595EE1 592E776D 3ED6A974 4D997CB3 DE9212C3 35ADF5C9 9916FD65 96562E89 4EAD0792 EBFC2712 2385F5B2 F34600FB D7BC20FB EB452A7B ECE1AA40
(i=7) v[16]	BE851B2D A85F6358 81E6FC3B 0BB28000 FA55A33A 87BE1FAD 4119370F 1E2261AA A1318FD3 F4329816 071783C2 6E536A8D 9A81A601 E7EC80F1 ACC09948 F849A584
(i=8) v[16]	07E5B85A 069CC164 F9DE3141 A56F4680 9E440AD2 9AB659EA 3C84B971 21DBD9CF 46699F8C 765257EC AF1D998C 75E4C3B6 523878DC 30715015 397FEE81 4F1FA799
(i=9) v[16]	435148C4 A5AA2D11 4B354173 D543BC9E BDA2591C BF1D2569 4FCB3120 707ADA48 565B3FDE 32C9C916 EAF4A1AB B1018F28 8078D978 68ADE4B5 9778FDA3 2863B92E
(i=10) v[16]	D9C994AA CFEC3AA6 700D0AB2 2C38670E AF6A1F66 1D023EF3 1D9EC27D 945357A5 3E9FFEBD 969FE811 EF485E21 A632797A DEEF082E AF3D80E1 4E86829B 4DEAFD3A
h[8]	8C5E8C50 E2147C32 A32BA7E1 2F45EB4E 208B4537 293AD69E 4C9B994D 82596786
BLAKE2s-256("abc")	50 8C 5E 8C 32 7C 14 E2 E1 A7 2B A3 4E EB 45 2F 37 45 8B 20 9E D6 3A 29 4D 99 9B 4C 86 67 59 82

4.2.2 Perancangan Sistem

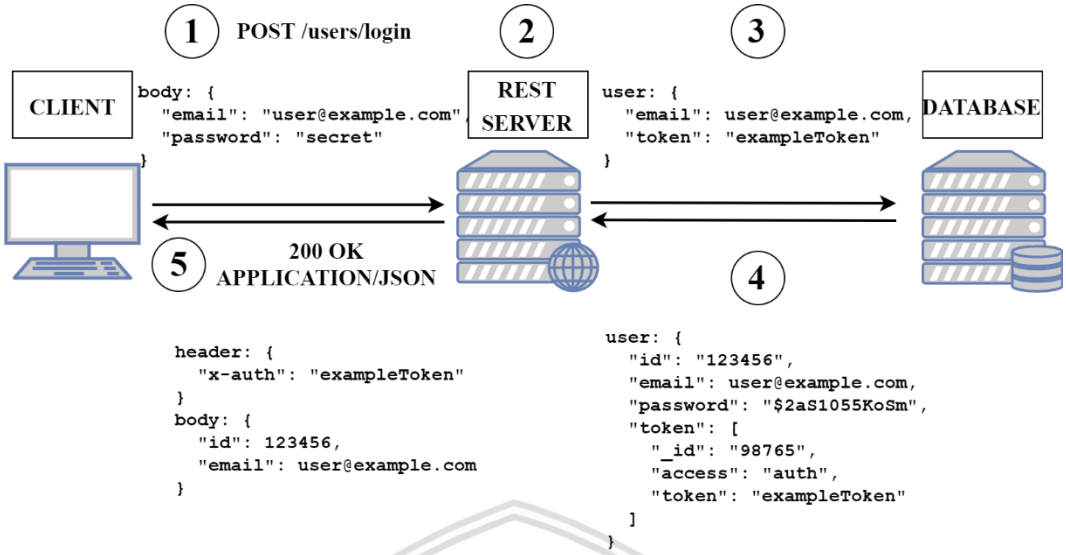
Dalam penelitian yang dilakukan yaitu perancangan sistem autentikasi REST API dengan JSON Web Token menggunakan algoritme BLAKE2S dibagi dalam 4 skema. Skema pertama yaitu *register* yang diilustrasikan Gambar 4.3, skema kedua yaitu *login* yang diilustrasikan Gambar 4.4, skema ketiga yaitu *user information* yang diilustrasikan Gambar 4.5 dan skema keempat yaitu *logout* yang diilustrasikan Gambar 4.6.



Gambar 4.3 Skema *Register* REST API.

Penjelasan skema *register* REST API Gambar 4.3 sebagai berikut

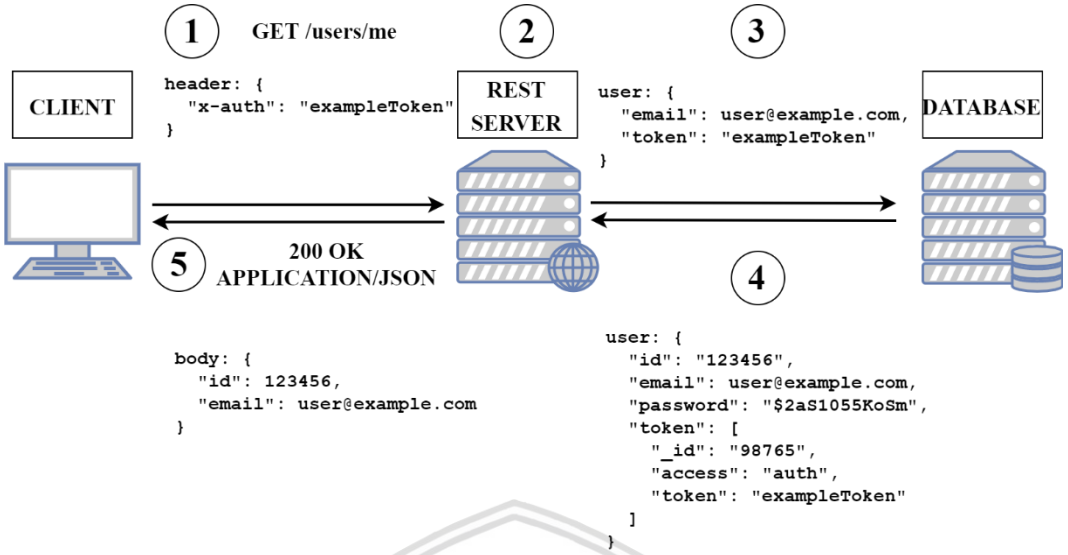
1. *Client* melakukan *register* dengan memasukan *email* dan *password* pada HTTP *body*.
2. *Server* REST melakukan pengecekan *email* agar tidak terjadi duplikasi, bila terjadi duplikasi *server* REST mengembalikan HTTP *status code* 400. Bila tidak terjadi dupilkasi maka dibuatkan sebuah token dan data *client* disimpan. Token dibuat berdasarkan nilai *key* yang sudah diberi statis pada *server*.
3. *Server* kemudian menyimpan data *email* berserta token pada *Database*.
4. *Database* mengembalikan data yang baru saja tersimpan kembali ke *server* REST.
5. *Server* REST membalas HTTP *request client* dengan *header* berisi token yang disematkan pada x-auth dan data publik *client* pada HTTP *body*.



Gambar 4.4 Skema *Login* REST API.

Penjelasan skema *register* REST API Gambar 4.4 sebagai berikut

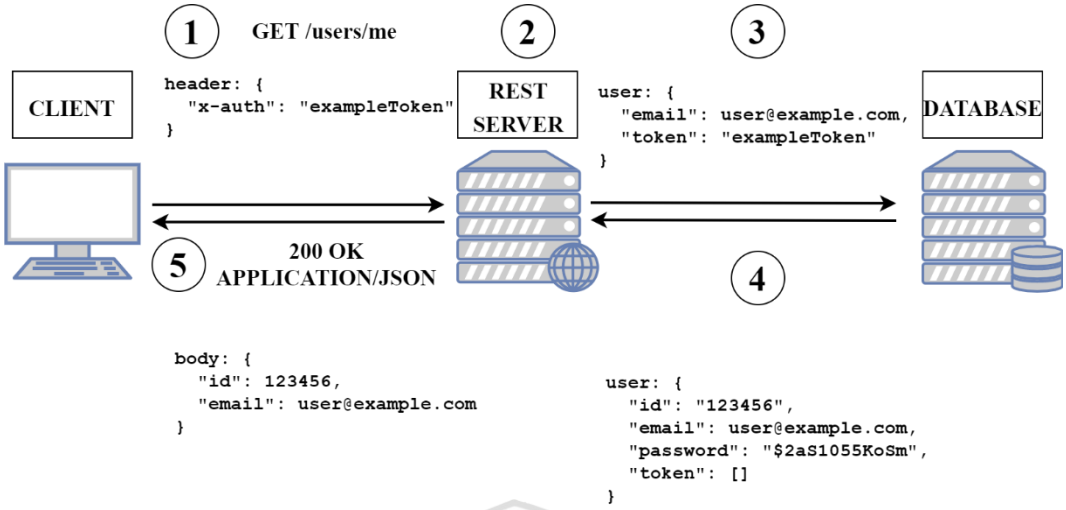
1. *Client* melakukan *login* dengan memasukan *email* dan *password* pada HTTP *Body*.
2. *Server* REST melakukan pengecekan *email* pada *Database* kemudian data *client* ditemukan maka server akan membuat token, apabila data *client* tidak ditemukan maka *server* REST akan mengembalikan HTTP status *code* 400. Token dibuat berdasarkan nilai *key* yang sudah diberi statis pada *server*.
3. Token yang dibuat disimpan berdasarkan email *client*.
4. *Database* akan mengembalikan data *client* kepada *server*.
5. *Server* REST membalas HTTP *request client* dengan *header* berisi token yang disematkan pada x-auth dan data publik *client* pada HTTP *body*.



Gambar 4.5 Skema *User Information* REST API.

Penjelasan skema *user information* REST API Gambar 4.5 sebagai berikut

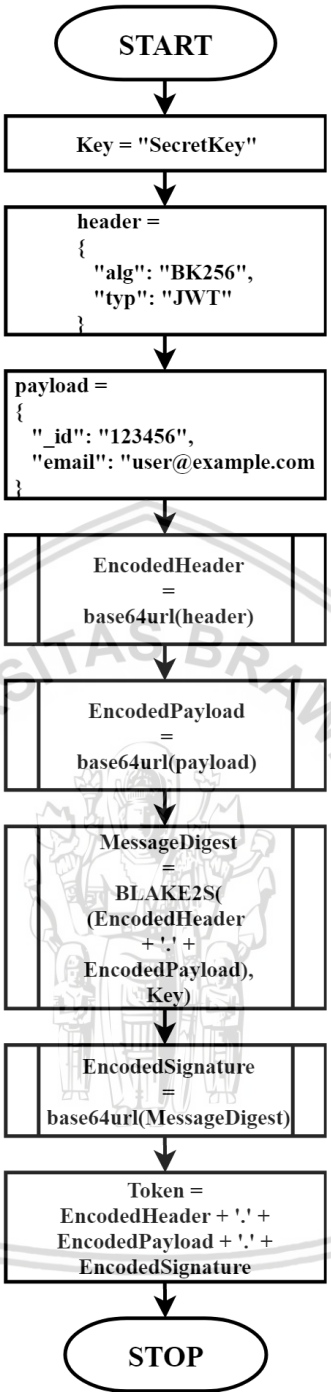
1. *Client* mengambil data pribadi dengan menyertakan token pada x-auth HTTP *header*.
2. *Server* REST melakukan pengecekan terhadap nilai token apabila nilai token *invalid* (tidak sesuai / telah dirubah) maka mengembalikan pesan HTTP status *code* 401.
3. *Server* REST melakukan pencarian data *client* berdasarkan kombinasi *email* dan token.
4. *Database* mengembalikan data pencarian pada *server* REST.
5. *Server* REST membalas HTTP *request client* dengan data publik *client* pada HTTP *body*.



Gambar 4.6 Skema *Logout* REST API.

Penjelasan skema *user information* REST API Gambar 4.6 sebagai berikut.

1. *Client* melakukan *logout* dengan menyertakan token pada x-auth HTTP *header*.
2. *Server* REST melakukan pengecekan terhadap nilai token apabila nilai token *invalid* (tidak sesuai / telah dirubah) maka mengembalikan pesan HTTP status *code* 401.
3. *Server* REST melakukan pencarian *client* yang memiliki nilai token yang sesuai dan menghapus token tersebut dari data *client* dalam *database*.
4. *Database* mengembalikan data pencarian pada *server* REST.
5. *Server* REST membalas HTTP *request* dengan status *code* 200.

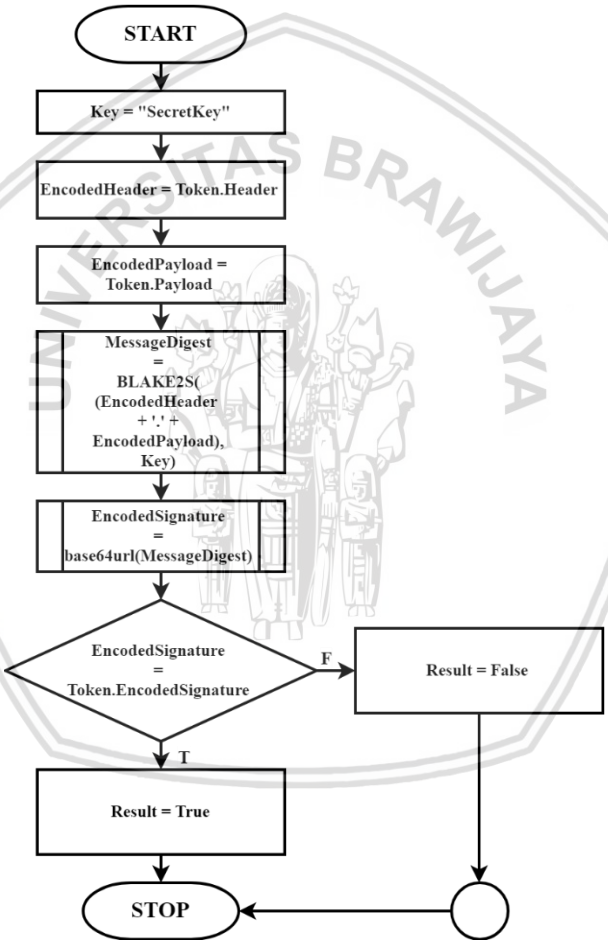


Gambar 4.7 Skema Pembuatan JSON Web Token.

Penjelasan skema dari pembuatan JSON Web Token Gambar 4.7 sebagai berikut.

1. Melakukan inisialisasi *key* yang sudah diberi nilai statis.
2. Membuat objek *header* dengan isi typ dan alg pada aturan JWT yang sudah diatur.
3. Memasukkan data *user* berupa *id* dan *email* kedalam objek *payload.*

4. *Header* JSON Web Token dirubah dalam bentuk *encoding* base64Url.
5. *Payload* JSON Web Token dirubah dalam bentuk *encoding* base64Url.
6. Pembuatan *signature* diawali dengan melakukan fungsi *hash* dimana dalam penelitian ini fungsi yang digunakan ialah BLAKE2S. Pesan yang akan dilakukan *hashing* dengan format "*encodedHeader*.*encodedPayload*" dengan menggunakan kunci rahasia yaitu *key*.
7. Hasil dari *message digest* dirubah dalam bentuk *encoding* base64Url.
8. Nilai token disusun dari "*encodedHeader*.*encodedPayload*. *encodedSignature* ".



Gambar 4.8 Skema Verifikasi JSON Web Token.

Penjelasan skema dari verifikasi JSON Web Token Gambar 4.8 sebagai berikut.

1. Mengambil nilai dari *key* pada nilai yang sudah diberi statis di awal.
2. Mengambil nilai *header* dari token yang didapat kemudian disimpan pada variabel EncodedHeader.

3. Mengambil nilai *payload* dari token yang didapat kemudian disimpan pada variabel EncodedPayload.
4. Token yang akan diverifikasi akan diambil nilai EncodedHeader dan EncodedPayload untuk dilakukan *hashing* dengan menggunakan *key*.
5. Hasil dari fungsi *hash* dirubah dalam bentuk base64url.
6. Kemudian dicocokan apakah nilai EncodedSignature yang baru sesuai dengan nilai *signature* token yang diterima. Bila benar maka token tersebut terverifikasi bila tidak maka token tersebut dianggap tidak valid.

Dengan adanya skema diatas, diharapkan tidak ada kemungkinan terjadinya upaya pelemahan tingkat keamanan oleh pihak yang tidak berwenang dari sisi aplikasi pada saat mengakses sumberdaya pada *server* REST API.

4.3 Perancangan Pengujian

Terdapat empat skenario pengujian yang dilakukan yaitu skenario *test vector*, skenario pengujian fungsional sistem, pengujian waktu autentikasi, dan pengujian keamanan.

4.3.1 Pengujian *Test Vector*

Pengujian *test vector* digunakan untuk menguji hasil *message digest* fungsi *hash* yang dihasilkan agar sesuai dengan hasil *output* fungsi *hash*. Pengujian ini menentukan apakah algoritme yang dibangun sesuai dengan algoritme yang ditentukan pada dokumen RFC 7693.

Pada pengujian ini dilakukan dengan mencocokan sebuah pesan contoh hasil *message digest* yang dihasilkan pada algoritme yang dibangun kemudian mencocokan dengan dokumen *test vector* yang disediakan pada dokumen resmi.

4.3.2 Pengujian Fungsional Sistem

Pengujian fungsional sistem digunakan untuk menguji apakah kebutuhan fungsional benar sesuai tujuan dan menghasilkan hasil sesuai tujuan dari kebutuhan fungsional.

Pengujian fungsional sistem dilakukan dengan *Integration testing function* yang kemudian melihat hasil pengujian apakah terdapat kesalahan atau tidak dalam menjalakan pengujian fungsional sistem.

4.3.3 Pengujian Waktu autentikasi

Pengujian autentikasi digunakan untuk mengetahui perbedaan dari algoritme yang digunakan dalam menghasilkan token yang digunakan autentikasi.

Pengujian waktu autentikasi dilakukan dengan melakukan autentikasi berulang dengan jumlah percobaan yang telah ditentukan.

BAB 5 IMPLEMENTASI

Implementasi merupakan tahap penerapan sistem yang telah dirancang pada tahap perancangan dan analisis sebelumnya. Dalam penelitian ini penulis menggunakan bahasa pemrograman NodeJS yang merupakan bahasa yang berasal dari bahasa Javascript. Berdasarkan hasil analisis dan perancangan yang sudah dibuat penulis melakukan implementasi algoritme BLAKE2S pada JSON Web Token sebagai algoritme *hash* untuk mekanisme layanan autentikasi REST API. Pengujian dilakukan dengan 4 skema yaitu pengujian *test vector* untuk menguji hasil *message digest* dari algoritme BLAKE2S yang sudah dibuat, pengujian fungsional sistem untuk menguji kebutuhan fungsional sistem berjalan sesuai dengan tujuan sistem, dan pengujian waktu autentikasi untuk mengetahui perbedaan waktu autentikasi terhadap algoritme yang digunakan, dan pengujian keamanan untuk mengetahui perbedaan penggunaan JWT pada REST. Pada tahap implementasi dibutuhkan perangkat lunak dan perangkat keras yang spesifikasinya sesuai dengan Tabel 5.1

Tabel 5.1 Spesifikasi Perangkat Lunak dan Keras.

No	Jenis Perangkat	Komponen
1	Perangkat Keras	a. Intel Core i7-4720HQ CPU @ 2.60GHz. b. RAM 8.00 GB DDR3. c. Harddisk dengan kapasitas ukuran 1 TB 7200RPM.
2	Perangkat Lunak	a. Sistem Operasi Ubuntu 18.04 Kernel v4.16.5 b. NodeJS v8.11.1 c. Postman v6.0.10 d. JSON Web Token v8.2.1 e. ExpressJS v4.16.3 f. MongoDB v3.0.6

5.1 Implementasi

Proses implementasi dilakukan dengan menerapkan algoritme BLAKE2S yang telah penulis buat sebelumnya berdasarkan perancangan dan perangkat lunak JSON Web Token menggunakan *library* yang diunduh pada situs https://github.com/auth0/node-jsonwebtoken. Pembuatan REST *server* dengan perangkat lunak ExpressJS yang berjalan pada *localhost port* 3000 dan menggunakan *library* JSON Web Token untuk REST *server* melakukan autentikasi dan dilakukan pemasangan algoritme BLAKE2S pada JSON Web Token.

5.1.1 Algoritme BLAKE2S

5.1.1.1 *Class Constructor* BLAKE2S

Class constructor pada Tabel 5.2 dan Tabel 5.3 digunakan untuk melakukan inisialisasi nilai konstan IV, konstan *message permutation* (sigma), dan ctx. Ctx atau *context* pada c*onstruct*or digunakan untuk menyimpan variabel yang digunakan saat melakukan komputasi agar memudahkan dalam pengambilan variabel. Ctx berupa objek yang menampung variabel

1. h = *Hash state*
2. b = *input buffer*
3. v = *vector workspace*
4. m = *message block*
5. c = *pointer input buffer*
6. t = *total number of byte*
7. outlen = panjang *output*
8. keylen = panjang *key*

variabel *hash state* digunakan untuk menyimpan nilai *hash state* awal yang diinisialisasi dengan nilai IV, *input buffer* digunakan untuk menyimpan *message* yang diinputkan, *vector workspace* digunakan untuk menyimpan nilai komputasi saat menjalankan fungsi *compress*, *message block* digunakan untuk menyimpan pesan blok yang berasal dari *input buffer* dalam bentuk *little endian*, *pointer* digunakan untuk mengetahui posisi terakhir dari *input buffer*, t digunakan untuk menghitung total nilai *byte*, outlen menyimpan nilai panjang keluaran fungsi *hash*, dan keylen digunakan untuk menyimpan panjang *key* yang digunakan.

Tabel 5.2 Fungsi *Constructor* BLAKE2S.

Algoritme 1: Fungsi Constructor	
1	constructor() {
2	this.IV = new Uint32Array([
3	0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A,
4	0x510E527F, 0x9B05688C, 0x1F83D9AB, 0x5BE0CD19,
5	]);
6	this.SIGMA = new Uint8Array([
7	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
8	14, 10, 4, 8, 9, 15, 13, 6, 1, 12, 0, 2, 11, 7, 5, 3,
9	11, 8, 12, 0, 5, 2, 15, 13, 10, 14, 3, 6, 7, 1, 9, 4,
10	7, 9, 3, 1, 13, 12, 11, 14, 2, 6, 5, 10, 4, 0, 15, 8,
11	9, 0, 5, 7, 2, 4, 10, 15, 14, 1, 11, 12, 6, 8, 3, 13,
12	2, 12, 6, 10, 0, 11, 8, 3, 4, 13, 7, 5, 15, 14, 1, 9,
13	12, 5, 1, 15, 14, 13, 4, 10, 0, 7, 6, 3, 9, 2, 8, 11,
14	13, 11, 7, 14, 12, 1, 3, 9, 5, 0, 15, 4, 8, 6, 2, 10,
15	6, 15, 14, 9, 11, 3, 0, 8, 12, 2, 13, 7, 1, 4, 10, 5,
16	10, 2, 8, 4, 7, 6, 1, 5, 15, 11, 9, 14, 3, 12, 13, 0

Tabel 5.3 Lanjutan Fungsi *Constructor* BLAKE2S.

Algoritme 1: Fungsi Constructor	
17	]);
18	this.ctx = {
19	h: new Uint32Array(this.IV),
20	b: new Uint32Array(64),
21	v: new Uint32Array(16),
22	m: new Uint32Array(16),
23	c: 0,
24	t: 0,
25	outlen : 32,
26	keylen : 0,
27	}
28	}

5.1.1.2 Fungsi ConvertUint32ToHex

Fungsi ini digunakan untuk merubah nilai masukan parameter berupa integer $2^{32} - 1$ menjadi bentuk heksadesimal. Implementasi dari fungsi convertUint32ToHex ditampilkan pada Tabel 5.4.

Tabel 5.4 Fungsi ConvertUint32ToHex BLAKE2S

Algoritme 2: Fungsi ConvertUint32ToHex	
1	convertUint32ToHex(bytes) {
2	const result = (0x100000000 + bytes).toString(16).substring(1);
3	return result;
4	}

5.1.1.3 Fungsi DebugPrintUint32

Fungsi ini digunakan untuk menampilkan hasil manualisasi *input* dimana sebelum dan sesudah sebuah proses hasil pengolahan ditampilkan pada konsole atau *terminal*. Fungsi ini menggunakan fungsi convertUint32ToHex untuk merubah nilai integer menjadi nilai heksadesimal. Implementasi fungsi debugPrintUint32 ditampilkan pada Tabel 5.5 dan Tabel 5.6.

Tabel 5.5 Fungsi DebugPrintUint32 BLAKE2S.

Algoritme 3: Fungsi DebugPrintUint32	
1	debugPrintUint32(label, array) {
2	let result = '\n' + label + ' = '
3	for (let i = 0; i < array.length; i += 2) {
4	result += this.convertUint32ToHex(array[i]).toUpperCase() +

Tabel 5.6 Lanjutan Fungsi DebugPrintUint32 BLAKE2S.

Algoritme 3: Fungsi ConvertUint32ToHex	
5	' ' +
6	this.convertUint32ToHex(array[i + 1]).toUpperCase();
7	if(i % 6 === 4) {
8	result += '\n\t' + new Array(label.length + 4).join(' ');
9	} else if (i < array.length - 2) {
10	result += ' ';
11	}
12	}
13	console.log(result);
14	}

5.1.1.4 Fungsi NormalizeInput

Fungsi normalizeInput digunakan untuk merubah masukan pesan yang diinputkan pada fungsi *hash* BLAKE2S untuk dirubah menjadi integer $2^8 - 1$ dalam bentuk *array*. Fungsi ini menerima 2 masukan yaitu dalam bentuk *string* dan juga dalam bentuk *array* integer $2^8 - 1$. Implementasi fungsi normalizeInput ditampilkan pada Tabel 5.7.

Tabel 5.7 Fungsi NormalizeInput BLAKE2S.

Algoritme 4: Fungsi NormalizeInput	
1	normalizeInput(input) {
2	if(typeof input === 'string'){
3	return new Uint8Array(Buffer.from(input, 'utf8'));
4	} else if(input instanceof Uint8Array) {
5	return input
6	} else {
7	throw new Error('type of input must a string!');
8	}
9	}

5.1.1.5 Fungsi ConvertToHex

Fungsi convertToHex digunakan untuk merubah integer $2^8 - 1$ menjadi bentuk heksadesimal. Fungsi ini menerima masukan berupa *array* integer $2^8 - 1$ dan Implementasi fungsi convertToHex ditampilkan pada Tabel 5.8.

Tabel 5.8 Fungsi ConvertToHex BLAKE2S.

Algoritme 5: Fungsi ConvertToHex	
1	convertToHex(bytes) {
2	const result = Array.prototype.map.call(bytes, function (n) {
3	return (n < 16 ? '0' : '') + n.toString(16)
4	}).join('');
5	return result;
6	}

5.1.1.6 Fungsi CheckOutputLength

Fungsi checkOutputLength digunakan untuk melakukan pengecekan terhadap masukan keluaran fungsi *hash* yang diinputkan oleh pengguna. Apabila masukan bernilai valid sesuai dengan aturan algoritme maka nilai keluaran disimpan dalam ctx. Implementasi fungsi checkOutputLength pada Tabel 5.9.

Tabel 5.9 Fungsi CheckOutputLength BLAKE2S.

Algoritme 6: Fungsi CheckOutputLength	
1	checkOutputLength(outlen) {
2	if(!(outlen > 0 && outlen <= 32)) {
3	throw new Error('Invalid output length');
4	}
5	this.ctx.outlen = outlen;
6	}

5.1.1.7 Fungsi CheckKeyLength

Fungsi checkKeyLength digunakan untuk melakukan pengecekan terhadap panjang *key* yang digunakan oleh fungsi *hash*. Apabila panjang nilai *key* tidak valid maka mengembalikan nilai *error*. Implementasi fungsi checkKeyLength pada Tabel 5.10.

Tabel 5.10 Fungsi CheckKeyLength BLAKE2S.

Algortime 7: Fungsi CheckKeyLength	
1	checkKeyLength(key) {
2	if(key && !(key.length > 0 && key.length <= 32)) {
3	throw new Error('Invalid key input lenght');
4	}
5	this.ctx.keylen = key ? key.length : 0;
6	}

5.1.1.8 Fungsi GetLittleEndianWord

Fungsi getLittleEndianWord digunakan untuk mengambil nilai 4 bit dari *input buffer* (b) kemudian dirubah dalam bentuk integer $2^8 - 1$ *little endian*. Fungsi getLittleEndianWord ditampilkan pada Tabel 5.11.

Tabel 5.11 Fungsi GetLittleEndianWord BLAKE2S.

Algoritme 8: Fungsi GetLittleEndianWord	
1	getLittleEndianWord(i) {
2	return this.ctx.b[i] ^ (this.ctx.b[i + 1] << 8) ^
3	(this.ctx.b[i + 2] << 16) ^ (this.ctx.b[i + 3] << 24)
4	}

5.1.1.9 Fungsi *Rotation*

Fungsi *rotation* digunakan untuk merotasi nilai bit sejumlah parameter y. Fungsi ini digunakan pada saat menjalankan G *function* dimana salah satu operasinya adalah *rotation*. Implementasi fungsi *rotation* pada Tabel 5.12.

Tabel 5.12 Fungsi *Rotation* BLAKE2S.

Algoritme 9: Fungsi Rotation	
1	rotation(x, y) {
2	return (x >>> y) ^ (x << (32 - y))
3	}

5.1.1.10 Fungsi *Update*

Fungsi *update* digunakan untuk mengisi nilai variabel *input buffer* pada ctx yang nilainya berasal dari masukan *message* dari *input has*h, apabila *nilai input buffer* sudah 512 *bit* atau nilai variabel ctx c bernilai 64 maka dilakukan komputasi ctx pada fungsi *compress*. Pada fungsi *compress* terdapat parameter *flag* yang digunakan untuk menandakan apakah pesan tersebut merupakan pesan terakhir atau tidak. Implementasi fungsi *update* ditampilkan pada Tabel 5.13.

Tabel 5.13 Fungsi *Update* BLAKE2S.

Algoritme 10: Fungsi Update	
1	update(input) {
2	for(let i = 0 ; i < input.length ; i++) {
3	if(this.ctx.c === 64) {
4	this.ctx.t += this.ctx.c;
5	this.compress(false);
6	this.ctx.c = 0;
7	}
8	this.ctx.b[this.ctx.c++] = input[i];
9	}
10	}

5.1.1.11 Fungsi G *Function*

G *Function* merupakan fungsi dasar bagi algoritme BLAKE2S. Fungsi ini digunakan untuk melakukan komputasi dimana didalamnya terdapat operasi *rotation*, *addition*, dan XOR. Nilai yang akan dikomputasi dalam G *function* ini adalah variable v pada ctx dimana perhitungan bergantung pada parameter G *function* berupa *message permutation* (SIGMA) dan masukan a, b, c, dan d. Implementasi G function ditampilkan pada Tabel 5.14.

Tabel 5.14 Fungsi G *Function* BLAKE2S.

Algoritme 11: Fungsi G	
1	gFunction(a, b, c, d, x, y) {
2	this.ctx.v[a] = this.ctx.v[a] + this.ctx.v[b] + x;
3	this.ctx.v[d] = this.rotation(this.ctx.v[d] ^ this.ctx.v[a], 16);
4	this.ctx.v[c] = this.ctx.v[c] + this.ctx.v[d];
5	this.ctx.v[b] = this.rotation(this.ctx.v[b] ^ this.ctx.v[c], 12);
6	this.ctx.v[a] = this.ctx.v[a] + this.ctx.v[b] + y;
7	this.ctx.v[d] = this.rotation(this.ctx.v[d] ^ this.ctx.v[a], 8);
8	this.ctx.v[c] = this.ctx.v[c] + this.ctx.v[d];
9	this.ctx.v[b] = this.rotation(this.ctx.v[b] ^ this.ctx.v[c], 7);
10	}

5.1.1.12 Fungsi *Compress*

Fungsi *compress* merupakan fungsi utama algoritme BLAKE2S dimana didalamnya melakukan komputasi untuk menghasilkan *message digest*. Penjelasan sebagai berikut

1. Baris 2 – 11: Inisialisasi *vector workspac*e untuk digunakan dalam komputasi fungsi G *function*.
2. Baris 12 – 14: Pengambilan nilai *little endian* dari *input blo*ck (ctx.b) dan disimpan kedalam *message block* (ctx.m).
3. Baris 15 – 56: Melakukan komputasi G *function* sejumlah 10 putaran dimana 1 kali putaran melakukan komputasi 8 kali G *function*. Masukan untuk G *function* berupa nilai yang sudah ditentukan oleh algoritme, *message block* yang ditentukan dari *message permutation* (SIGMA).
4. Baris 57 – 60: melakukan *finalization* dimana hasil komputasi dari *vector workspace* dilakukan operasi XOR dengan *hash state* untuk menghasilkan *final hash state* yang akan disambung menjadi *message digest*.

Implementasi fungsi *compress* ditampilkan pada Tabel 5.15, dan Tabel 5.16.

Tabel 5.15 Fungsi *Compress* BLAKE2S.

Algoritme 12: Fungsi Compress

```
1   compress(lastFlag) {
2       let i;
3       for(i = 0 ; i < 8 ; i++) {
4         this.ctx.v[i] = this.ctx.h[i];
5         this.ctx.v[i + 8] = this.IV[i];
6       }
7       this.ctx.v[12]^= this.ctx.t;
8       this.ctx.v[13]^= (this.ctx.t / 0x100000000);
9       if(lastFlag) {
10        this.ctx.v[14] = ~this.ctx.v[14]
11      }
12      for(i = 0; i < 16 ; i++) {
13        this.ctx.m[i] = this.getLittleEndianWord(4 * i);
14      }
15      for(i = 0; i < 10; i++) {
16        this.gFunction(
17          0, 4, 8, 12,
18          this.ctx.m[this.SIGMA[i * 16 + 0]],
19          this.ctx.m[this.SIGMA[i * 16 + 1]],
20        );
21        this.gFunction(
22          1, 5, 9, 13,
23          this.ctx.m[this.SIGMA[i * 16 + 2]],
24          this.ctx.m[this.SIGMA[i * 16 + 3]]
25        );
26        this.gFunction(
27          2, 6, 10, 14,
28          this.ctx.m[this.SIGMA[i * 16 + 4]],
29          this.ctx.m[this.SIGMA[i * 16 + 5]]
30        );
31        this.gFunction(
32          3, 7, 11, 15,
33          this.ctx.m[this.SIGMA[i * 16 + 6]],
34          this.ctx.m[this.SIGMA[i * 16 + 7]]
35        );
36        this.gFunction(
37          0, 5, 10, 15,
38          this.ctx.m[this.SIGMA[i * 16 + 8]],
39          this.ctx.m[this.SIGMA[i * 16 + 9]]
```

Tabel 5.16 Lanjutan Kedua Fungsi *Compress* BLAKE2S.

Algoritme 12: Fungsi Compress	
40	); this.gFunction(1, 6, 11, 12, this.ctx.m[this.SIGMA[i * 16 + 10]], this.ctx.m[this.SIGMA[i * 16 + 11]]
41	
42	
43	
44	
45	); this.gFunction(2, 7, 8, 13, this.ctx.m[this.SIGMA[i * 16 + 12]], this.ctx.m[this.SIGMA[i * 16 + 13]]
46	
47	
48	
49	
50	); this.gFunction(3, 4, 9, 14, this.ctx.m[this.SIGMA[i * 16 + 14]], this.ctx.m[this.SIGMA[i * 16 + 15]]
51	
52	
53	
54	
55	); }
56	
57	for (i = 0; i < 8; i++) {
58	this.ctx.h[i] ^= this.ctx.v[i] ^ this.ctx.v[i + 8]
59	}
60	}

5.1.1.13 Fungsi *Final*

Fungsi final merupakan fungsi yang dijalankan apabila *input block* merupakan 512 bit terakhir. Fungsi ini akan menjalanan fungsi *compress* dimana akan memperbarui nilai ctx kemudian nilai *hash state* ctx (h) akan dirubah menjadi heksadesimal dari integer *little endia*n. Penjelasan sebagai berikut

1. Baris 2 – 5: pengisian bit sisa dengan 0 agar panjang pesan sejumlah 512 bit.
2. Baris 6: melakukan fungsi *compress* dengan *flag final true*.
3. Baris 8: inisialisasi *array* integer $2^8 - 1$ dengan panjang sesuai nilai keluaran yang diinginkan.
4. Baris 9 – 11 : Pengambilan nilai dari *hash stat*e (ctx.h) dalam bentuk integer *little endian* dan dirubah menjadi *integer*.

Implementasi fungsi final ditampilkan pada Tabel 5.17.

Tabel 5.17 Fungsi Final BLAKE2S.

Algoritme 13: Fungsi Final	
1	final() {
2	this.ctx.t += this.ctx.c;
3	while (this.ctx.c < 64) {
4	this.ctx.b[this.ctx.c++] = 0;
5	}
6	this.compress(true);
7	
8	const raw = new Uint8Array(this.ctx.outlen);
9	for (var i = 0; i < this.ctx.outlen; i++) {
10	raw[i] = (this.ctx.h[i >> 2] >> (8 * (i & 3))) & 0xFF;
11	}
12	return raw;
13	}

5.1.1.14 Fungsi *Hash*

Fungsi *hash* merupakan fungsi utama dalam melakukan komputasi *hash* algoritme BLAKE2S. Fungsi ini digunakan untuk melakukan pengecekan dan validasi masukan algoritme BLAKE2S yang kemudian menentukan jenis hasil keluaran dari fungsi *hash*. Penjelasan sebagai berikut

1. Baris 2: melakukan pengecekan dan validasi panjang keluaran fungsi *hash*.
2. Baris 3: melakukan pengecekan dan validasi panjang *key* yang digunakan.
3. Baris 5 – 6: merubah masukan berupa *string* menjadi *array* integer $2^8 - 1$.
4. Baris 8: menggabungkan panjang nilai *key*, panjang keluaran pada *hash state* (ctx.h) dengan operasi XOR.
5. Baris 10 – 13: melakukan *update context* (ctx) apabila menggunakan kunci untuk melakukan komputasi *hash*.
6. Baris 15: melakukan *update context* (ctx) dengan masukan pesan (dataInput).
7. Baris 16: mengambil nilai dari *hash state final* (ctx.h) dan disimpan dalam variabel raw.
8. Baris 19 – 32: menentukan jenis keluaran fungsi *hash* berdasarkan parameter masukan pada fungsi *hash*. Nilai masukan untuk jenis keluaran berdasarkan nilai dari variabel raw. Nilai keluaran yang bisa digunakan adalah heksadesimal dan base64. Keluaran base64 digunakan untuk masukan proses dari JSON Web Token.

Implementasi fungsi *hash* ditampilkan pada Tabel 5.18.

Tabel 5.18 Fungsi *Hash* BLAKE2S.

Algoritme 14: Fungsi Hash

```
1    hash(input, key, length, outputType) {
2      this.checkOutputLength(length);
3      this.checkKeyLength(key);
4
5      const dataInput = this.normalizeInput(input);
6      const keyInput = this.normalizeInput(key);
7
8      this.ctx.h[0] ^= 0x01010000 ^ (this.ctx.keylen << 8) ^
   this.ctx.outlen;
9
10     if(this.ctx.keylen > 0) {
11       this.update(keyInput);
12       this.ctx.c = 64;
13     }
14
15     this.update(dataInput);
16     const raw = this.final();
17
18     let result;
19     switch (outputType) {
20       case 'hex' : {
21         result = this.convertToHex(raw);
22         break;
23       }
24       case 'base64' : {
25         result = ab2str(raw, 'base64');
26         break;
27         }
28       default : {
29         console.log(outputType);
30         throw new Error('Output type is invalid');
31       }
32     }
33     return result;
34   }
```

5.1.1.15 Fungsi DoHash

Fungsi doHash merupakan fungsi *static* dimana tidak perlu melakukan instansiasi objek dari kelas BLAKE2S. Fungsi ini digunakan untuk menginstansiasi

objek dari kelas BLAKE2S untuk melakukan komputasi *hash*. Fungsi doHash ditampilkan pada Tabel 5.19.

Tabel 5.19 Fungsi DoHash BLAKE2S.

Algoritme 15: Fungsi DoHash	
1	static doHash(input = '', key = '', length = 32, outputType = 'hex') {
2	const blake2s = new Blake2s();
3	const result = blake2s.hash(input, key, length, outputType);
4	return result;
5	}

5.1.2 *Server* REST API

5.1.2.1 *File* Config.js

File ini digunakan untuk memuat nilai variabel yang disimpan dalam variabel proses ketika progam berjalan. Variabel yang dimuat berasal dari *file* config.json dimana menyimpan variabel PORT, MONGODB_URI, dan JWT_SECRET. Implementasi dari *file* config.js ditampilkan pada Tabel 5.20 dan implementasi file config.json ditampilkan pada Tabel 5.20, Tabel 5.21, dan Tabel 5.22.

Tabel 5.20 *File* Config.js.

Algoritme 16: File Config	
1	const env = process.env.NODE_ENV || 'development';
2	
3	if (env === 'test' || env === 'development') {
4	const config = require('./config.json');
5	const envConfig = config[env];
6	
7	Object.keys(envConfig).forEach((key) => {
8	process.env[key] = envConfig[key];
9	})
10	}

Tabel 5.21 *File* Config.json.

Algoritme 17: File Config JSON	
1	{
2	"test" : {
3	"PORT" : 3000,
4	"MONGODB_URI" : "mongodb://127.0.0.1:27017/DbTest",
5	"JWT_SECRET" : "b741aa12"
6	},
7	"development" : {

Tabel 5.22 Lanjutan File Config.json.

Algoritme 17: File Config JSON	
8	"PORT" : 3000,
9	"MONGODB_URI" : "mongodb://127.0.0.1:27017/DbDevel",
10	"JWT_SECRET" : "b741aa12"
12	}
13	}

5.1.2.2 *File* Db.js

File Db.js digunakan untuk melakukan koneksi pada *database* MongoDB. *File* ini menggunakan API mongoose untuk melakukan koneksi dan transaksi pada *database* MongoDB. Variable mongoose kemudian di *export* untuk digunakan pada *file* lain. Implementasi dari *file* db.js ditampilkan pada Tabel 5.23.

Tabel 5.23 *File* Db.js.

Algoritme 18: File DB	
1	const mongoose = require('mongoose');
2	mongoose.Promise = global.Promise;
3	mongoose.connect(process.env.MONGODB_URI);
4	module.exports = {mongoose};

5.1.2.3 *File* Authenticate.js

File Authenticate.js merupakan *file* untuk melakukan autentikasi terhadap token JWT yang disematkan pada *header* HTTP. File ini bertindak sebagai *middleware* pada *resource* URL dari REST *server*. Token JWT didapat melalui *header* HTTP kemudian dilakukan pencarian pada *database* untuk mencari pemilik dari token tersebut, apabila tidak ditemukan maka REST *server* akan mengembalikan kode status 401 (*Unauthorized*) HTTP. Implementasi *file* authenticate.js ditampilkan pada Tabel 5.24 dan Tabel 5.25.

Tabel 5.24 *File* Authenticate.js.

Algoritme 19: File Authenticate	
1	const {User} = require('./../model/User');
2	
3	const authenticate = async (req, res, next) => {
4	const token = req.header('x-auth');
5	try{
6	const user = await User.findByToken(token);
7	if(!user){
8	throw new Error('User not found');
9	}

Tabel 5.25 Lanjutan File Authenticate.js.

Algoritme 19: File Authenticate	
10	
11	req.user = user;
12	req.token = token;
13	next();
14	}catch (e){
15	return res.status(401).send();
16	}
17	};
18	
19	module.exports = {
20	authenticate
21	};

5.1.2.4 *File* Model User.js

File User.js merupakan *file* untuk merepresentasikan model dari *user*. *File* ini merupakan *file* skema *user* untuk transaksi pada *database* dimana menyimpan model dari objek *user* dan fungsi untuk memanipulasi data dalam *database*. Pada *file* ini digunakan *library* JWT (JSON Web Token) untuk pembuatan token dan *library* bcrypt untuk melakukan *hashing password user* sebelum disimpan dalam *database*. Penjelasan kode sebagai berikut.

1. Baris 7 – 37: Merepresentasikan skema model *user* pada transaksi *database* sekaligus merepresentasikan model *user*. Terdapat beberapa attribut yang yaitu *email*, *password*, dan token. Setiap atribut dari model *user* memiliki aturan / format yang sudah ditentukan dalam JSON atribut tersebut.
2. Baris 39 – 43: Fungsi yang digunakan dalam setiap transaksi *database* dimana hasil dari *database* dirubah kedalam bentuk JSON dan mengambil atribut *id* dan *email*.
3. Baris 45 – 52: Fungsi untuk membuat token JWT untuk model *user*. Token JWT dibuat dari atribut *id user* model dan *access* berupa nilai *string* auth yang disematkan dalam *payload* JWT. Kemudian menggunakan *key* berupa nilai dari variabel JWT_SECRET. Setelah token dibuat maka token tersebut disimpan dalam *database* dan fungsi ini mengembalikan nilai token JWT.
4. Baris 54 – 63: Fungsi untuk menghapus token JWT pada *database*.
5. Baris 65 – 78: Fungsi statik dimana digunakan untuk mencari nilai token JWT pada *database*. Sebelum melakukan pencarian dilakukan pengecekan token JWT apakah token tersebut valid atau tidak dengan menggunakan nilai JWT_SECRET untuk melakukan validasi, apabila token ditemukan maka mengembalikan objek *user*.

6. Baris 80 – 94: Fungsi statik yang digunakan untuk melakukan pencarian *user* berdasarkan kombinasi *email* dan *password*. Pencarian dilakukan dengan menggunakan *email* masukan dan melakukan validasi *password* masukan dengan algoritme bcrypt, apabila kombinasi *email* dan *password* sesuai maka mengembalikan nilai objek *user*.
7. Baris 96 – 111: Fungsi yang dijalankan ketika menyimpan data *user* dalam *database*. Fungsi ini melakukan *hash* pada atribut *password user* agar disimpan dalam bentuk *message digest*.

Implementasi *file* model user.js ditampilkan pada Tabel 5.26, Tabel 5.27, Tabel 5.28, dan Tabel 5.29.

Tabel 5.26 *File* Model User.js.

Algoritme 20: File Model User	
1	`const mongoose = require('mongoose');`
2	`const validator = require('validator');`
3	`const jwt = require('jsonwebtoken');`
4	`const _ = require('lodash');`
5	`const bcrypt = require('bcryptjs');`
6	
7	`const UserSchema = new mongoose.Schema({`
8	`  email: {`
9	`    type: String,`
10	`    require: true,`
11	`    trim: true,`
12	`    minlength: 1,`
13	`    unique: true,`
14	`    validate: {`
15	`      validator: validator.isEmail,`
16	`      message: '{VALUE} is not a valid email!',`
17	`      isAsync: true`
18	`    }`
19	`  },`
20	`  password: {`
21	`    type: String,`
27	`    require: true,`
28	`    minlength: 6`
29	`  },`
30	`  tokens: [{`
31	`    access: {`
32	`      type: String,`
33	`      require: true`
34	`    },`

Tabel 5.27 Lanjutan Pertama *File* Model User.js.

Algoritme 20: File Model User	
35	token: {
36	type: String,
37	require: true
38	}
39	}]
40	}, {
41	usePushEach: true
42	});
43	
44	UserSchema.methods.toJSON = function () {
45	const User = this;
46	const userObject = User.toObject();
47	return _.pick(userObject, ['_id', 'email']);
48	};
49	UserSchema.methods.generateAuthToken = async function () {
50	const User = this;
51	const access = 'auth';
52	const token = await jwt.sign({_id: User._id.toHexString(),
53	access}, process.env.JWT_SECRET).toString();
54	User.tokens.push({access, token});
55	User.save();
56	return token;
57	};
58	
59	UserSchema.methods.removeToken = function (token) {
60	const User = this;
61	return User.update({
62	$pull : {
63	tokens : {
64	token
65	}
66	}
67	});
68	};
69	UserSchema.statics.findByToken = async function (token) {
70	const User = this;
71	try {
72	

Tabel 5.28 Lanjutan Kedua *File* Model User.js.

Algoritme 20: File Model User	
73	const decoded = await jwt.verify(token, process.env.JWT_SECRET);
74	return User.findOne({
75	'_id' : decoded._id,
76	'tokens.token' : token,
77	'tokens.access' : 'auth'
78	});
79	} catch (e) {
80	throw new Error(e);
81	}
82	};
83	
84	UserSchema.statics.findByCredentials = async function (email, password) {
85	const User = this;
86	
87	const user = await User.findOne({email});
88	if(!user){
89	throw new Error('User not found');
90	}
91	
92	const status = await bcrypt.compare(password, user.password);
93	if(status){
94	return user;
95	}else {
96	throw new Error('password doesn\'t match');
97	}
98	};
99	UserSchema.pre('save', async function (next) {
100	const User = this;
101	if(User.isModified('password')){
102	try{
103	const salt = await bcrypt.genSalt(10);
104	const pass = await bcrypt.hash(User.password, salt);
105	User.password = pass;
106	next();
107	}catch (e){
108	console.log(e);
109	}

Tabel 5.29 Lanjutan Ketiga *File* Model User.js.

Algoritme 20: File Model User	
110	}else{
111	next();
112	}
113	});
114	const User = mongoose.model('User', UserSchema);
115	module.exports = {User};

5.1.2.5 *File Route* User.js

File User.js merupakan representasi *resource* URL dari REST *server*. Terdapat 4 *resource* yaitu

1. POST /users: *Resource* yang digunakan untuk mendaftarkan *user* kedalam sistem. *Resource* ini menggunakan *method* HTTP POST untuk melewatkan data *user email* dan *password* pada HTTP *body*.
2. GET /users/me: *Resource* yang digunakan untuk melihat data *user* yang sedang *login* melalui token JWT yang disematkan pada *header* HTTP. *Resource* ini menggunakan *middleware* authenticate.js untuk melakukan validasi token yang dikirim pada *header* HTTP. Apabila nilai token tidak ada atau tidak valid maka pengguna tidak dapat melakukan permintaan pada *resource* ini.
3. POST /users/login: *Resource* yang digunakan untuk melakukan autentikasi berdasarkan kombinasi *email* dan *password*. *Resource* ini menggunakan *method* POST untuk melewatkan data *user* berupa *email* dan *password* pada HTTP *body*.
4. DELETE /users/logout: *Resource* yang digunakan untuk melakukan *logout* dari sistem dimana menggunakan *middleware* authenticate.js untuk melakukan validasi token yang disematkan pada HTTP *header*. Apabila nilai token valid maka token tersebut akan dihapus dari *database*.

Implementasi dari *file route* user.js ditampilkan pada Tabel 5.30, Tabel 5.31, dan Tabel 5.32.

Tabel 5.30 *File Route* User.js.

Algoritme 21: File Route User	
1	const express = require('express');
2	const {ObjectID} = require('mongodb').ObjectID;
3	const _ = require('lodash');
4	
5	const router = express.Router();
6	
7	const {User} = require('./../model/User');

Tabel 5.31 Lanjutan Pertama *File Route* User.js.

Algoritme 20: File Model User	
8	}else{
9	next();
10	}
11	});
12	const User = mongoose.model('User', UserSchema);
13	module.exports = {User};
14	require('./../config/Db');
15	const {authenticate} = require('./../middleware/Authenticate');
16	
17	router.post('/users', async (req, res) => {
18	const body = _.pick(req.body, ['email', 'password']);
19	const newUser = new User(body);
20	
21	try{
22	await newUser.save();
23	const userToken = await newUser.generateAuthToken();
24	return res.header('x-auth', userToken).send(newUser);
25	}catch (e){
26	return res.status(400).send(e);
28	}
29	});
30	
31	router.get('/users/me', authenticate, (req, res) => {
32	res.send(req.user);
33	});
34	router.post('/users/login', async (req, res) => {
35	const {email, password} = _.pick(req.body, ['email', 'password']);
36	try{
37	const user = await User.findByCredentials(email, password);
38	const userToken = await user.generateAuthToken();
39	res.header('x-auth', userToken).send(user);
40	}catch (e){
41	res.status(400).send(e);
42	}
43	});
44	router.delete('/users/logout',authenticate, async (req, res) => {
45	try{
46	await req.user.removeToken(req.token);

Tabel 5.32 Lanjutan Kedua *File Route* User.js.

Algoritme 21: File Route User	
47	res.status(200).send()
48	} catch(e){
49	res.status(400).send(e)
50	}
51	});
52	module.exports = router;

5.1.2.6 *File* App.js

File App.js merupakan file konfigurasi dan inisialisasi REST *server*. Penjelasan sebagai berikut

1. Baris 1: Memuat variabel proses dari REST *server.*
2. Baris 3: Memuat *library file system* untuk membaca *file* pada *server* REST.
3. Baris 4: Memuat *library* ExpressJS.
4. Baris 5: Memuat *library* body-parser.
5. Baris 9: Inisialisasi REST *server.*
6. Baris 10: Mengambil nilai variabel *port* dari proses variabel *config.*
7. Baris 12: Memuat konfigurasi dari *library* body-parser untuk digunakan pada REST *server*. *Library* tersebut digunakan untuk mengaktifkan HTTP *body* berupa bentuk JSON.
8. Baris 13: Memuat konfigurasi *resource* URL untuk dipasangkan pada REST *server*.
9. Baris 15-17: Menjalankan REST *server* pada nomor *port* yang nilainya disimpan dalam variabel PORT.
10. Baris 19 – 23: Fungi terminasi REST *server*.

Implementasi dari *file* app.js ditampilkan pada Tabel 5.33 dan Tabel 5.34.

Tabel 5.33 *File* App.js.

Algoritme 22: File App	
1	require('./config/Config');
2	
3	const fs = require('fs');
4	const express = require('express');
5	const bodyParser = require('body-parser');
6	
7	const users = require('./routes/User');
8	const app = express();
9	const port = process.env.PORT;
10	app.use(bodyParser.json());
11	app.use('/', users);

Tabel 5.34 Lanjutan *File* App.js.

Algoritme 22: File App	
12	
13	`app.listen(port, () => {`
14	`  console.log(`App is starting at port ${port}`);`
15	`});`
16	
17	`function stop () {`
18	`  app.listen(port, function() {`
19	`    app.close();`
20	`  });`
21	`}`
22	
23	`module.exports = {`
24	`  app,`
25	`  stop`
26	`};`
27	

5.1.3 JSON Web Token

5.1.3.1 *File* JSON Web Token Sign.js

File ini merupakan file pembuatan JWT dimana didalamnya terdapat konfigurasi mencakup algoritme yang dapat digunakan beserta JSON Web Signature (JWS). Beberapa konfigurasi dilakukan dapat menggunakan algoritme BLAKE2S sebagai algoritme *hash* dalam JWT. Pada baris 5 perlu ditambahkan nama algoritme BLAKE2S yaitu BK256 agar dapat dikenali. Pada baris 16 hingga 19 merupakan *header* dari JWT yang nilai algoritme nya diberikan nilai *default* yaitu BK256. Implementasi file sign.js pada Tabel 5.35 dan Tabel 5.36.

Tabel 5.35 *File* JSON Web Token Sign.js.

Algoritme 23: File JSON Web Token Sign	
1	`var sign_options_schema = {`
2	`  expiresIn: { isValid: function(value) { return isInteger(value) || isString(value); }, message: '"expiresIn" should be a number of seconds or string representing a timespan' },`
3	`  notBefore: { isValid: function(value) { return isInteger(value) || isString(value); }, message: '"notBefore" should be a number of seconds or string representing a timespan' },`

Tabel 5.36 Lanjutan *File* JSON Web Token Sign.js.

Algoritme 23: File JSON Web Token Sign	
4	audience: { isValid: function(value) { return isString(value) || Array.isArray(value); }, message: '"audience" must be a string or array' },
5	algorithm: { isValid: includes.bind(null, ['RS256', 'RS384', 'RS512', 'ES256', 'ES384', 'ES512', 'HS256', 'HS384', 'HS512', 'BK256', 'none']), message: '"algorithm" must be a valid string enum value' },
6	header: { isValid: isPlainObject, message: '"header" must be an object' },
7	encoding: { isValid: isString, message: '"encoding" must be a string' },
8	issuer: { isValid: isString, message: '"issuer" must be a string' },
9	subject: { isValid: isString, message: '"subject" must be a string' },
10	jwtid: { isValid: isString, message: '"jwtid" must be a string' },
11	noTimestamp: { isValid: isBoolean, message: '"noTimestamp" must be a boolean' },
12	keyid: { isValid: isString, message: '"keyid" must be a string' },
13	mutatePayload: { isValid: isBoolean, message: '"mutatePayload" must be a boolean' }
14	};
15	var header = xtend({
16	alg: options.algorithm || 'BK256',
17	typ: isObjectPayload ? 'JWT' : undefined,
18	kid: options.keyid
19	}, options.header);

5.1.3.2 *File* JSON Web Token Verify.js

File ini digunakan dalam melakukan verifikasi token JWT yang digunakan. Konfigurasi pada Tabel 5.37 dan Tabel 5.38 digunakan untuk mengenali algoritme BLAKE2S yang nilainya disematkan pada *header* JWT.

Tabel 5.37 *File* JSON Web Token Verify.js.

Algoritme 24: File JSON Web Token Verify	
1	if (!options.algorithms) {
2	options.algorithms = ~secretOrPublicKey.toString().indexOf('BEGIN CERTIFICATE') ||
3	~secretOrPublicKey.toString().indexOf('BEGIN PUBLIC KEY') ?

Tabel 5.38 Lanjutan *File* JSON Web Token Verify.js.

Algoritme 24: File JSON Web Token Verify	
	['RS256','RS384','RS512','ES256','ES384','ES512'] :
4	~secretOrPublicKey.toString().indexOf('BEGIN RSA PUBLIC KEY') ? ['RS256','RS384','RS512'] : ['HS256','HS384','HS512', 'BK256']; }

5.1.3.3 *File* JSON Web Signature Index.js

File ini digunakan dalam pembuatan JWT dimana terdapat algoritme yang dapat digunakan. Konfigurasi pada Tabel 5.39 digunakan untuk mengenali algoritme BLAKE2S pada JWT.

Tabel 5.39 *File* JSON Web Signature Index.js.

Algoritme 25: File JSON Web Signature	
1	var ALGORITHMS = [
2	'HS256', 'HS384', 'HS512',
3	'RS256', 'RS384', 'RS512',
4	'ES256', 'ES384', 'ES512',
5	'BK256',
6	];

5.1.3.4 *File* JSON Web Algorithm Index.js

File ini merupakan yang digunakan dalam proses pembuatan JWT termasuk pembuatan dan verifikasi token. Pada *file* ini perlu ditambahkan konfigurasi agar dapat menggunakan algoritme BLAKE2S, perubahan ditampilkan dalam Tabel 5.40 dan Tabel 5.41. penjelasan sebagai berikut

1. Baris 1: Menambahkan pesan *error* apabila algoritme yang digunakan tidak ditemukan.
2. Baris 2 – 10: Merupakan fungsi untuk membuat *signature* JWT. Nilai masukan untuk membuat *signature* berupa *header* dan *payload* JWT kemudian dilakukan *hashing*. Hasil dari fungsi *hash* algortime BLAKE2S berupa base64 yang kemudian dirubah dalam bentuk base64url.
3. Baris 11-16: Merupakan fungsi untuk melakukan verifikasi token JWT yang digunakan. Fungsi ini melakukan verifikasi dengan membuat token JWT dari *header* dan *payload* yang diterima kemudian hasilnya di cocokan dengan *signature* JWT yang diterima.
4. Baris 18 – 42: Merupakan fungsi utama dalam *file* index.js. Fungsi ini melakukan inisialisasi fungsi yang disimpan kedalam objek kemudian

melakukan pencarian algoritme mana yang harus digunakan saat membuat dan melakukan verifikasi token.

Tabel 5.40 *File* JSON Web Algorithm Index.js.

Algoritme 26: File JSON Web Algorithm	
1	var MSG_INVALID_ALGORITHM = '"%s" is not a valid algorithm.\n Supported algorithms are:\n "HS256", "HS384", "HS512", "RS256", "RS384", "RS512", "BK256"and "none".'
2	function createBlake2sSigner(bits) {
3	return function sign(thing, secret) {
4	if (!bufferOrString(secret))
5	throw typeError(MSG_INVALID_SECRET);
6	thing = normalizeInput(thing);
7	const raw = Blake2s.doHash(thing, secret, 32, 'base64');
8	return base64url.fromBase64(raw);
9	}
10	}
11	function createBlake2sVerifier(bits) {
12	return function verify(thing, signature, secret) {
13	var computedSig = createBlake2sSigner(bits)(thing, secret);
14	return bufferEqual(Buffer.from(signature), Buffer.from(computedSig));
15	}
16	}
17	module.exports = function jwa(algorithm) {
18	var signerFactories = {
19	bk: createBlake2sSigner,
20	hs: createHmacSigner,
21	rs: createKeySigner,
22	es: createECDSASigner,
23	none: createNoneSigner,
24	}
25	var verifierFactories = {
26	bk: createBlake2sVerifier,
27	hs: createHmacVerifier,
28	rs: createKeyVerifier,
29	es: createECDSAVerifer,
30	none: createNoneVerifier,
31	}
32	var match = algorithm.match(/^(RS|ES|HS|BK)(256|384|512)$\|^(none)$/i);
34	if (!match)

Tabel 5.41 Lanjutan *File* JSON Web Algorithm Index.js.

Algoritme 26: File JSON Web Algorithm	
35	throw typeError(MSG_INVALID_ALGORITHM, algorithm);
36	var algo = (match[1] || match[3]).toLowerCase();
37	var bits = match[2];
38	
39	return {
40	sign: signerFactories[algo](bits),
41	verify: verifierFactories[algo](bits),
42	}
43	};

BAB 6 PENGUJIAN DAN HASIL ANALISIS

Bab Pengujian dan analisis bertujuan untuk membahas pengujian yang dilakukan setelah tahap implementasi dilakukan juga analisis dari hasil implementasi yang sudah dilakukan.

6.1 Parameter Pengujian

Pada penelitian ini menggunakan beberapa parameter pengujian untuk menguji algoritme dan sistem REST API yang sudah dibangun. Parameter pengujian sebagai berikut

1. Pengujian *test vector* : pengujian validitas algoritme BLAKE2S dengan *test vector* yang terdapat pada dokumen RFC 7693.
2. Pengujian fungsionalitas sistem : pengujian fungsionalitas sistem dengan *integration function testing*.
3. Pengujian waktu autentikasi : Pengujian waktu autentikasi terhadap sistem REST API dengan algoritme BLAKE2S dan algoritme HMAC-SHA256.

6.2 Pengujian *Test Vector*

Pengujian *test vector* dilakukan agar memastikan bahwa algoritme yang sudah dibuat pada tahap implementasi memiliki hasil *message digest* yang sama seperti algoritme yang telah dibuat oleh pembuatnya pada dokumen RFC 7693.

6.2.1 Prosedur Pengujian

Prosedur pengujian *test vector* dapat dilakukan dengan cara memberikan masukan yang telah ditentukan pada dokumen *test vector* RFC 7693 algoritme BLAKE2S kemudian membandingkan hasil keluaran berupa *message digest* dengan *message digest* yang ada pada dokumen *test vector*. Format yang digunakan dalam pengujian ini berupa masukan pesan *string* dan keluaran *message digest* berupa heksadesimal.

6.2.2 Hasil dan Analisis

6.2.2.1 Masukan *String* Tanpa *Key*

1. *Message* : abc

   Key : ‘’ (kosong)

   Output : 508c5e8c327c14e2e1a72ba34eeb452f37458b209ed63a294d999b4c86675982

Dari percobaan pengujian *test vector* algoritme BLAKE2S hasil output *message digest* bernilai sama dengan nilai *test vector* yang disediakan algoritme BLAKE2S, sehingga implementasi algoritme BLAKE2S yang telah dibuat bersifat valid dan benar.

6.3 Pengujian Fungsionalitas Sistem

Pengujian fungsionalitas sistem dilakukan untuk memastikan bahwa fungsionalitas sistem REST API yang dibangun dapat berjalan tanpa menemukan kendala atau *error*. Pada pengujian ini dilakukan dengan *integration testing* yang menggunakan *library* jest dimana setiap *resource* URL dari REST API akan diuji.

6.3.1 Prosedur Pengujian

Sebelum melakukan pengujian diperlukan data contoh untuk digunakan setiap *integration testing*. Pada Tabel 6.1 Terdapat 2 buah data contoh yang digunakan dimana setiap data memiliki atribut *id*, *password* dan token.

Tabel 6.1 Tabel Data Contoh Pengujian.

Algoritme 27: Data Contoh Pengujian Fungsional Sistem

```
1   const dummyUser = [{
2     _id : idUserOne,
3     email : 'userOne@examples.com',
4     password : 'secretOne',
5     tokens : [{
6       access : 'auth',
7       token : jwt.sign({_id: idUserOne.toHexString(), access : 'auth'}, process.env.JWT_SECRET).toString()
      }]
8   },{
9     _id : idUserTwo,
10    email : 'userTwo@examples.com',
11    password : 'secretTwo',
12    tokens : [{
13      access : 'auth',
14      token : jwt.sign({_id: idUserTwo.toHexString(), access : 'auth'}, process.env.JWT_SECRET).toString()
15    }]
16  }];
```

Setiap pengujian akan dilakukan pembuatan data contoh diatas maka tidak ada ketergantungan data setiap pengujian yang akan dilakukan.

6.3.2 Hasil dan Analisis

6.3.2.1 *GET* /users/me

Pada *resource GET* /users/me dilakukan 2 skema percobaan yaitu.

1. Mengembalikan data *user* apabila melakukan autentikasi pada Tabel 6.2.
2. Mengembalikan status *code* 401 apabila belum melakukan autentikasi pada Tabel 6.3.

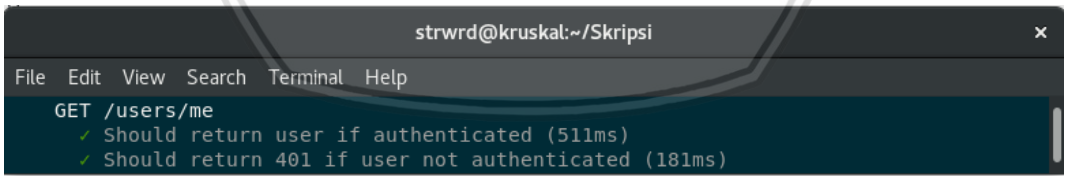
Tabel 6.2 Percobaan Pertama *GET* /users/me.

Algoritme 28: Pengujian Pertama GET /users/me	
1	`it('Should return user if authenticated', async () => {`
2	`const result = await request(app)`
3	`.get('/users/me')`
4	`.set('x-auth', dummyUser[0].tokens[0].token)`
5	`.expect(200);`
6	`expect(result.body.email).toBe(dummyUser[0].email);`
7	`expect(result.body._id).toBe(dummyUser[0]._id.toHexString());`
8	`});`

Tabel 6.3 Percobaan Kedua *GET* /users/me.

Algoritme 29: Pengujian Kedua GET /users/me	
1	`it('Should return user if authenticated', async () => {`
2	`const result = await request(app)`
3	`.get('/users/me')`
4	`.set('x-auth', dummyUser[0].tokens[0].token)`
5	`.expect(200);`
6	`expect(result.body.email).toBe(dummyUser[0].email);`
7	`expect(result.body._id).toBe(dummyUser[0]._id.toHexString());`
8	`});`

Setelah percobaan dilakukan maka didapatkan hasil sesuai dengan ekspektasi nilai yang diharapkan untuk *GET* /users/me yang ditampilkan pada Gambar 6.1.



Gambar 6.1 Hasil Pengujian *GET* /users/me.

6.3.2.2 *POST* /users

Pada percobaan *POST* /users dilakukan 3 percobaan yaitu

1. Dapat melakukan registrasi pada Tabel 6.4.
2. Tidak dapat melakukan registrasi apabila nilai *email* terjadi duplikasi pada Tabel 6.5.
3. Mengembalikan *error* apabila data yang diinputkan bernilai tidak valid pada Tabel 6.6.

Tabel 6.4 Percobaan Pertama *POST* /users.

Algoritme 30: Pengujian Pertama POST /users

```
1     it('Should create a new user', async () => {
2
3       const email = 'userThree@examples.com';
4       const password = 'secretThree';
5
6       const result = await request(app)
7         .post('/users')
8         .send({email, password})
9         .expect(200);
10
11      expect(result.body.email).toBe(email);
12      expect(result.body._id).toBeTruthy();
13      expect(result.header['x-auth']).toBeTruthy();
14
15      const obj = await User.findOne({email});
16      if (!obj) {
17        throw new Error('user not found');
18      }
19      expect(obj).toBeTruthy();
20      expect(obj.password).not.toBe(password);
21    });
```

Tabel 6.5 Percobaan Kedua *POST* /users.

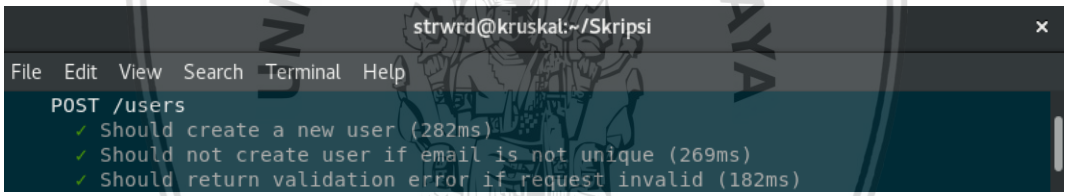
Algoritme 31: Pengujian Kedua POST /users

```
1   it('Should not create user if email is not unique', async () => {
2       const email = 'userOne@examples.com';
3       const password = 'secretOne';
4
5       await request(app)
6         .post('/users')
7         .send({email, password})
8         .expect(400);
9     });
```

Tabel 6.6 Percobaan Ketiga *POST* /users.

Algoritme 32: Pengujian Ketiga POST /users	
1	`it('Should return validation error if request invalid', async () => {`
2	`const email = '';`
3	`const password = 'adfgfx';`
4	
5	`await request(app)`
6	`.post('/users')`
7	`.send({email, password})`
8	`.expect(400);`
9	`});`

Dari ketiga percobaan yang dilakukan didapatkan hasil sesuai dengan ekspektasi nilai yang diharapkan. Hasil dari ketiga percobaan tersebut ditampilkan pada Gambar 6.2.



Gambar 6.2 Hasil Pengujian *POST* /users.

6.3.2.3 *POST* /users/login

Pada pengujian *POST* /users/login dilakukan 2 pengujian yaitu

1. Mengembalikan token JWT apabila data kredensial yang dimasukan benar dan valid dengan data yang tersimpan dalam *database* pada Tabel 6.7.
2. Mengembalikan *error* dan token tidak diberikan apabila data yang dimasukan tidak valid atau data *user* tidak ada dalam *database* pada Tabel 6.8.

Tabel 6.7 Percobaan Pertama *POST* /users/login.

Algoritme 33: Pengujian Pertama POST /users/login

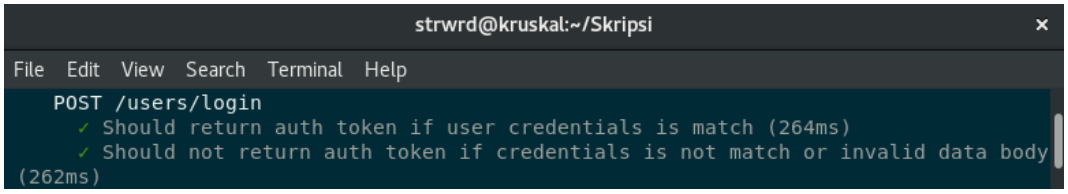
```
1   it('Should return auth token if user credentials is match', async () => {
2         const {email, password} = _.pick(dummyUser[1], ['email', 'password']);
4         const result = await request(app)
5           .post('/users/login')
6           .send({email, password})
7           .expect(200);
8         expect(result.header['x-auth']).toBeTruthy();
9         const obj = await User.findById(dummyUser[1]._id);
10        expect(obj.toObject().tokens[1]).toMatchObject({
11          access: 'auth',
12          token: result.header['x-auth']
13        });
```

Tabel 6.8 Percobaan Kedua *POST* /users/login.

Algoritme 34: Pengujian Kedua POST /users/login

```
1   it('Should not return auth token if credentials is not match or invalid data body', async () => {
2         const result = await request(app)
4           .post('/users/login')
5           .send({
6             email: 'userTwo@examples.com',
7             password: 'secretTow'
8           })
9           .expect(400);
10
11        expect(result.header['x-auth']).toBeFalsy();
12        const obj = await User.findById(dummyUser[1]._id);
13        expect(obj.tokens.length).toBe(1);
14      });
```

Dari kedua percobaan yang dilakukan didapatkan hasil nilai sesuai ekspektasi yang ditampilkan pada Gambar 6.3.



Gambar 6.3 Hasil Pengujian *POST* /users/login.

6.3.2.4 *DELETE* /users/logout

Pada pengujian *DELETE* /users/logout terdapat 2 pengujian yaitu

1. Menghapus token dalam *database* apabila token pengguna ditemukan pada Tabel 6.9.
2. Mengembalikan status *code* 401 apabila token pengguna yang digunakan tidak valid pada Tabel 6.10.

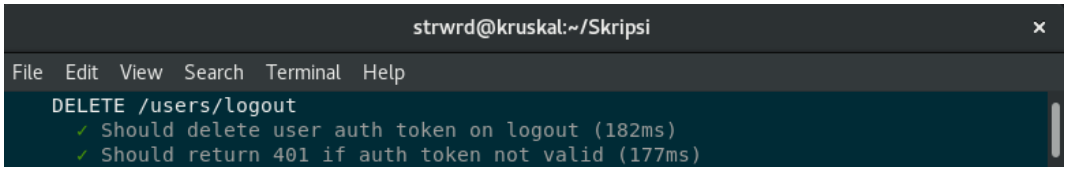
Tabel 6.9 Percobaan Pertama *DELETE* /users/login.

Algoritme 35: Pengujian Pertama DELETE /users/login	
1	`it('Should delete user auth token on logout', async () => {`
2	`await request(app)`
4	`.delete('/users/logout')`
5	`.set('x-auth', dummyUser[0].tokens[0].token)`
6	`.expect(200);`
7	`const obj = await User.findById(dummyUser[0]._id);`
8	`expect(obj.tokens.length).toBe(0);`
9	`});`

Tabel 6.10 Percobaan Kedua *DELETE* /users/login.

Algoritme 36: Pengujia Kedua DELETE /users/login	
1	`it('Should return 401 if auth token not valid', async () => {`
2	`await request(app)`
3	`.delete('/users/logout')`
4	`.set('x-auth', '')`
5	`.expect(401);`
6	`});`

Dari kedua percobaan yang dilakukan hasil yang didapatkan sesuai dengan nilai ekspektasi yang ditampilkan pada Gambar 6.4.



Gambar 6.4 Hasil Percobaan *DELETE* /users/logout.

6.3.2.5 Pengujian Verifikasi Token JWT Bernilai Valid

Pengujian verifikasi token JWT dilakukan 3 kali percobaan untuk memastikan bahwa token yang dihasilkan bernilai valid. Ketiga percobaan tersebut dilakukan sesuai dengan Tabel 6.11, Tabel 6.12, dan Tabel 6.13.

Tabel 6.11 Percobaan Pertama Verifikasi Token Valid.

Algoritme 37: Pengujian Pertama Verifikasi JWT Valid	
1	`it(`Signature Verified 1`,() => {`
2	`const token = jwt.sign({data: "using blake2s as hash algorithm"}, 'secret');`
3	`let result;`
4	`try {`
5	`const decoded = jwt.verify(token, 'secret');`
6	`result = true;`
7	`} catch (e) {`
8	`result = false;`
9	`}`
10	`expect(result).toBeTruthy();`
11	`});`

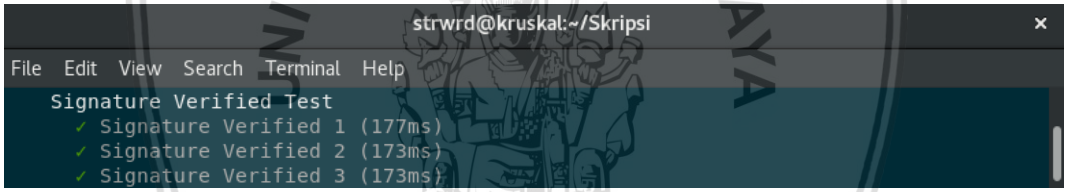
Tabel 6.12 Percobaan Kedua Verifikasi Token Valid.

Algoritme 38: Pengujian Kedua Verifikasi JWT Valid	
1	`it(`Signature Verified 2`,() => {`
2	`const token = jwt.sign({data: "blake2s is amazingly fast"}, 'thisIsSecret');`
3	`let result;`
4	`try {`
5	`const decoded = jwt.verify(token, 'thisIsSecret');`
6	`result = true;`
7	`} catch (e) {`
8	`result = false;`
9	`}`
10	`expect(result).toBeTruthy();`
11	`});`

Tabel 6.13 Percobaan Ketiga Verifikasi Token JWT Valid.

Algoritme 39: Pengujian Ketiga Verifikasi JWT Valid	
1	`it(`Signature Verified 3`,() => {`
2	`const token = jwt.sign({data: "blake2 is faster than MD5"}, 'privateKey');`
3	`let result;`
4	`try {`
5	`const decoded = jwt.verify(token, 'privateKey');`
6	`result = true;`
7	`} catch (e) {`
8	`result = false;`
9	`}`
10	`expect(result).toBeTruthy();`
11	`});`

Dari ketiga percobaan yang dilakukan untuk pengujian validitas JWT bernilai valid sesuai dengan ekspektasi nilai yang diharapkan. Hasil percobaan ditampilkan pada Gambar 6.5.



Gambar 6.5 Hasil Percobaan Verifikasi Token JWT Valid.

6.3.2.6 Pengujian Verifikasi Token JWT Bernilai Tidak Valid

Pengujian verifikasi token yang bernilai tidak valid dilakukan sebanyak 4 kali dengan beberapa skenario yang berbeda. Pengujian sebagai berikut

1. Token JWT bernilai tidak valid apabila nilai *signature* dari JWT diubah pada Tabel 6.14.
2. Token JWT bernilai tidak valid apabila *key* JWT bernilai tidak sama (nilai *key* yang diinputkan tidak sesuai dengan *key* asli) pada Tabel 6.15.
3. Token JWT bernilai tidak valid apabila *key* JWT bernilai tidak sama (nilai *key* sama tetapi terdapat perbedaan huruf kapital) pada Tabel 6.16.
4. Token JWT bernilai tidak valid apabila *key* JWT bernilai tidak sama (nilai *key* terdapat angka pada huruf vokal) pada Tabel 6.17.

Tabel 6.14 Percobaan Pertama Verifikasi Token JWT Tidak Valid.

Algoritme 40: Pengujian Pertama Verifikasi JWT Tidak Valid	
1	`it('Signature Invalid when token is tempered', () => {`
2	`const token = jwt.sign({data: "using blake2s as hash algorithm"}, 'secret');`
3	`const temperedToken = token.slice(0, -3)+'abc';`
4	`let result;`
5	`try {`
6	`const decoded = jwt.verify(temperedToken, 'secret');`
7	`result = true;`
8	`} catch (e) {`
9	`result = false;`
10	`}`
11	`expect(result).toBeFalsy();`
12	`});`

Tabel 6.15 Percobaan Kedua Verifikasi Token JWT Tidak Valid.

Algoritme 41: Pengujian Kedua Verifikasi JWT Tidak Valid	
1	`it('Signature Invalid when key doesn\'t match 1', () => {`
2	`const token = jwt.sign({data: "using blake2s as hash algorithm"}, 'secret');`
3	`let result;`
4	`try {`
5	`const decoded = jwt.verify(token, 'secreto');`
6	`result = true;`
7	`} catch (e) {`
8	`result = false;`
9	`}`
10	`expect(result).toBeFalsy();`
11	`});`

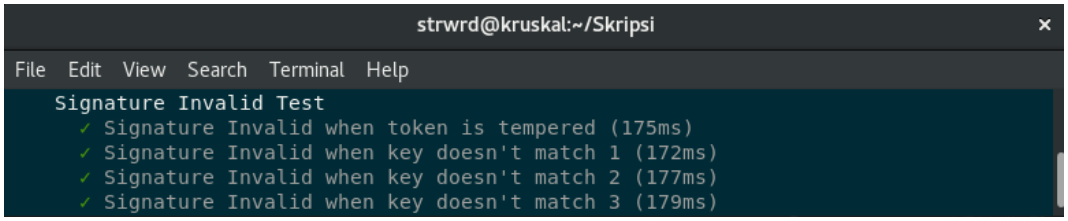
Tabel 6.16 Percobaan Ketiga Token JWT Bernilai Tidak Valid.

Algoritme 42: Pengujian Ketiga Verifikasi Token Tidak Valid	
1	`it('Signature Invalid when key doesn\'t match 2', () => {`
2	`const token = jwt.sign({data: "using blake2s as hash algorithm"}, 'secret');`
3	`let result;`
4	`try {`
5	`const decoded = jwt.verify(token, 'Secret');`
6	`result = true;`
7	`} catch (e) {`
8	`result = false;`
9	`}`
10	`expect(result).toBeFalsy();`
11	`});`

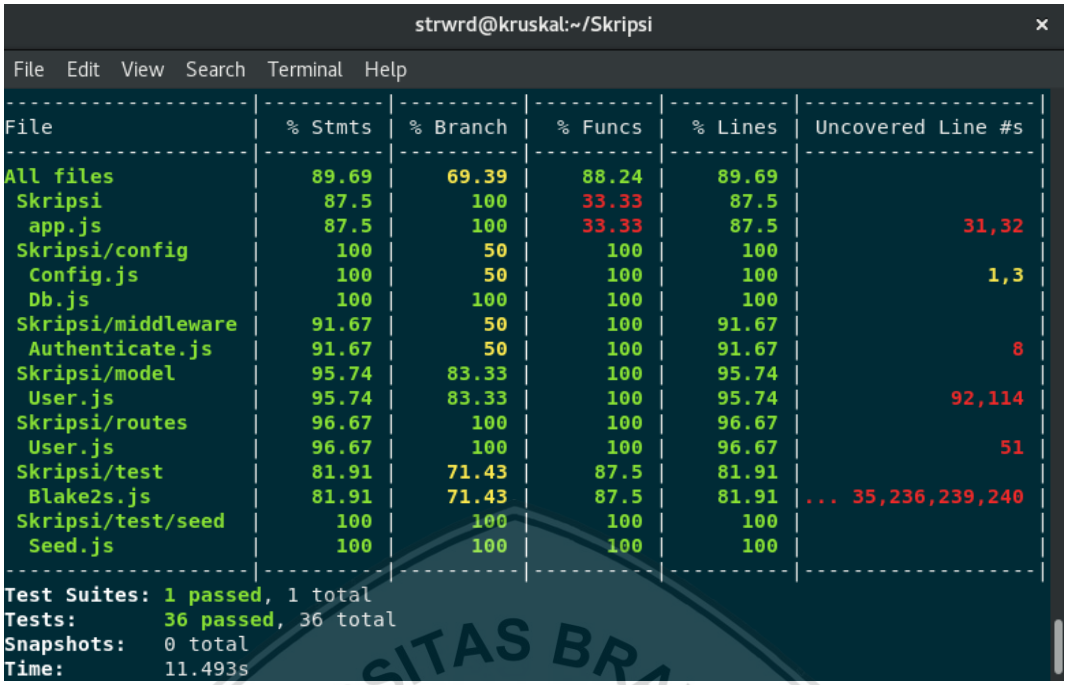
Tabel 6.17 Percobaan Keempat Token JWT Bernilai Tidak Valid.

Algoritme 43: Pengujian Keempat Verifikasi Token Tidak Valid	
1	`it('Signature Invalid when key doesn\'t match 3', () => {`
2	`const token = jwt.sign({data: "using blake2s as hash algorithm"}, 'secret');`
3	`let result;`
4	`try {`
5	`const decoded = jwt.verify(token, 's3cr3t');`
6	`result = true;`
7	`} catch (e) {`
8	`result = false;`
9	`}`
10	`expect(result).toBeFalsy();`
11	`});`

Dari keempat percobaan yang dilakukan tidak ditemukan kesalahan dalam pengujian dalam arti pengujian dilakukan dengan benar dan nilai yang didapatkan sesuai pada ekspektasi yang ditampilkan pada Gambar 6.6.



Gambar 6.6 Hasil Percobaan Token JWT Bernilai Tidak Valid.



Gambar 6.7 Hasil Keseluruhan Percobaan Fungsionalitas Sistem REST API

Dari 36 percobaan fungsionalitas termasuk 20 percobaan *test vector* yang dilakukan dengan *integration testing* tidak ditemukan kesalahan dan bernilai sesuai dengan ekspektasi percobaan maka dapat disimpulkan bahwa sistem yang dibangun dapat berjalan dengan baik tanpa menemukan kendala atau *error*. Hasil seluruh percobaan ditampilkan pada Gambar 6.7.

6.4 Pengujian Waktu Autentikasi

Pengujian waktu autentikasi merupakan pengujian yang dilakukan untuk mengetahui dan membandingkan waktu yang digunakan saat melakukan autentikasi dengan menggunakan algoritme BLAKE2S maupun HMAC-SHA256.

6.4.1 Prosedur Pengujian

Prosedur pengujian waktu autentikasi dilakukan dengan membuat data contoh *user* terlebih dahulu untuk digunakan autentikasi pada *resource* URL *POST* /users/login. Setelah membuat data contoh kemudian dilakukan autentikasi sebanyak 30 kali percobaan pada masing masing algoritme yaitu BLAKE2S dengan HMAC-SHA256 untuk mengetahui tiap percobaan berapa waktu yang dibutuhkan. Pengujian ini masing masing pada algoritme dilakukan sebanyak 30 dengan skenario yang sama. Menurut Hair et al. (2010) ketika jumlah sample sejumlah 30 atau lebih terdapat nilai normalitas pada sample tersebut maka nilai normalitas tersebut akan mempengaruhi hasil dari pengujian. Pada akhir percobaan disimpan nilai keseluruhan percobaan dan rata rata waktu yang dibutuhkan kemudian hasilnya akan ditampilkan. Dalam percobaan pada algoritme setiap proses autentikasi berjalan secara sekuensial dalam arti proses autentikasi berjalan satu per satu. Skema Percobaan dilakukan sesuai dengan pengujian pada Tabel 6.18, Tabel 6.19, dan Tabel 6.20.

6.4.2 Hasil dan Analisis

Sebelum menjalankan pengujian waktu autentikasi perlu dilakukan penambahan data *user* kedalam *database* agar dapat langsung melakukan autentikasi pada *resource* URL *POST* /users/login. Data atribut dari model *user* yang dimasukkan ke dalam database ditampilkan pada Gambar 6.8.

Gambar 6.8 Memasukan Data Contoh pada *Database* MongoDB.

Tabel 6.18 *File* Pengujian Waktu Autentikasi.

Algoritme 44: Pengujian Waktu Autentikasi	
1	`const time = require('performance-now');`
2	`const {app, stop} = require('../app');`
3	`const axios = require('axios');`
4	
5	`async function doLogin(attempt, email, password) {`
6	`  let t0;`
7	`  let t1;`
8	`  let result;`
9	`  let xAuth;`
10	`  try {`
11	`    t0 = time();`
12	`    result = await axios.post('http://127.0.0.1:3000/users/login', {`
13	`      email,`
14	`      password,`
15	`    });`
16	`    t1 = time();`
17	`    xAuth = result.headers['x-auth'];`
18	`  } catch (e) {`

Tabel 6.19 Lanjutan Pertama *File* Pengujian Waktu Autentikasi.

Baris	*Source Code*
19	`    console.log(`error attempt ${attempt}: ${e}`);`
20	`  }`
21	`  await axios.delete('http://127.0.0.1:3000/users/logout', {`
22	`    headers: {`
23	`      "x-auth": xAuth,`
24	`    }`
25	`  });`
26	`  console.log(`time ${attempt}: ${t1 - t0}ms`);`
27	`  return t1-t0;`
28	`}`
29	`async function timeTest () {`
30	`  let totalTime = 0;`
31	`  console.log('Start testing');`
32	`  await doLogin(1, 'time@test.com', 'time123');`
33	`  await doLogin(2, 'time@test.com', 'time123');`
34	`  await doLogin(3, 'time@test.com', 'time123');`
35	`  totalTime += await doLogin(4, 'time@test.com', 'time123');`
36	`  totalTime += await doLogin(5, 'time@test.com', 'time123');`
37	`  totalTime += await doLogin(6, 'time@test.com', 'time123');`
38	`  totalTime += await doLogin(7, 'time@test.com', 'time123');`
39	`  totalTime += await doLogin(8, 'time@test.com', 'time123');`
40	`  totalTime += await doLogin(9, 'time@test.com', 'time123');`
41	`  totalTime += await doLogin(10, 'time@test.com', 'time123');`
42	`  totalTime += await doLogin(11, 'time@test.com', 'time123');`
43	`  totalTime += await doLogin(12, 'time@test.com', 'time123');`
44	`  totalTime += await doLogin(13, 'time@test.com', 'time123');`
45	`  totalTime += await doLogin(14, 'time@test.com', 'time123');`
46	`  totalTime += await doLogin(15, 'time@test.com', 'time123');`
47	`  totalTime += await doLogin(16, 'time@test.com', 'time123');`
48	`  totalTime += await doLogin(17, 'time@test.com', 'time123');`
49	`  totalTime += await doLogin(18, 'time@test.com', 'time123');`
50	`  totalTime += await doLogin(19, 'time@test.com', 'time123');`
51	`  totalTime += await doLogin(20, 'time@test.com', 'time123');`
52	`  totalTime += await doLogin(21, 'time@test.com', 'time123');`
53	`  totalTime += await doLogin(22, 'time@test.com', 'time123');`
54	`  totalTime += await doLogin(23, 'time@test.com', 'time123');`
55	`  totalTime += await doLogin(24, 'time@test.com', 'time123');`
56	`  totalTime += await doLogin(25, 'time@test.com', 'time123');`
57	`  totalTime += await doLogin(26, 'time@test.com', 'time123');`

Tabel 6.20 Lanjutan Kedua *File* Pengujian Waktu Autentikasi.

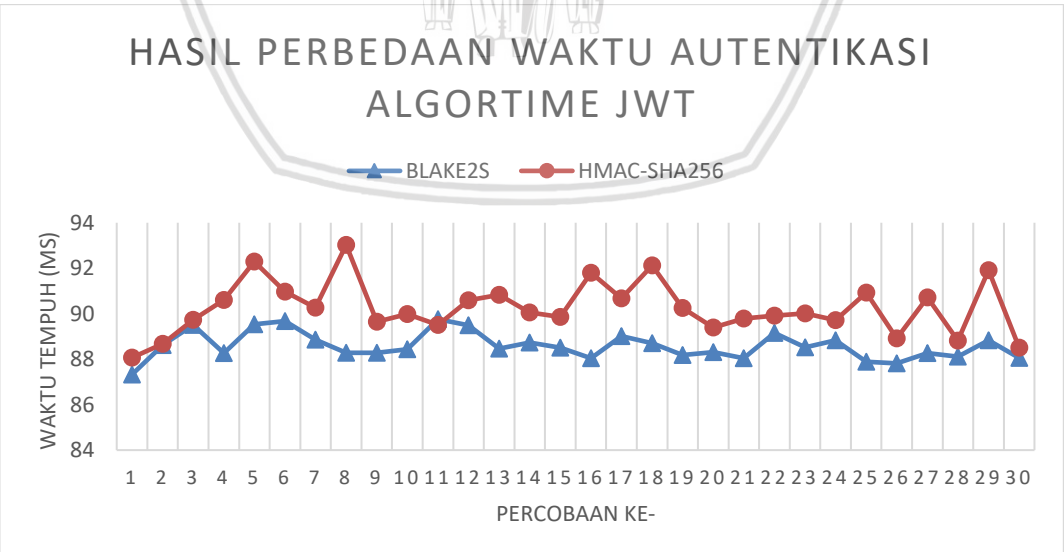
Algoritme 44: Pengujian Waktu Autentikasi	
58	`  totalTime += await doLogin(27, 'time@test.com', 'time123');`
59	`  totalTime += await doLogin(28, 'time@test.com', 'time123');`
60	`  totalTime += await doLogin(29, 'time@test.com', 'time123');`
61	`  totalTime += await doLogin(30, 'time@test.com', 'time123');`
62	`  totalTime += await doLogin(31, 'time@test.com', 'time123');`
63	`  totalTime += await doLogin(32, 'time@test.com', 'time123');`
64	`  totalTime += await doLogin(33, 'time@test.com', 'time123');`
65	`  console.log(`Total time: ${totalTime}`);`
66	`  console.log(`Average time: ${totalTime / 30}`);`
67	`}`
68	`timeTest().then(() => console.log('Done'));`

Tabel 6.21 Hasil Pengujian Waktu Autentikasi.

Percobaan Ke-	BLAKE2S (ms)	HMAC-SHA256 (ms)
1	87.32585899999992	88.06821000000002
2	88.62348599999996	88.67216099999996
3	89.50549000000001	89.73659900000007
4	88.26203600000008	90.59963000000005
5	89.5387609999998	92.29656099999988
6	89.67520299999978	90.97405299999991
7	88.8560829999999	90.265533
8	88.27892599999996	93.02283200000011
9	88.27669300000002	89.65019200000006
10	88.43054500000017	89.98477099999991
11	89.75727000000006	89.5083619999998
12	89.48789699999998	90.59264299999995
13	88.4695479999998	90.82824199999982
14	88.72796099999982	90.04932099999996
15	88.51103100000023	89.86272200000008
16	88.03540699999985	91.80119000000013
17	89.01173699999981	90.66909299999998

Tabel 6.22 Lanjutan Hasil Pengujian Waktu Autentikasi.

Percobaan Ke-	BLAKE2S (ms)	HMAC-SHA256 (ms)
18	88.70819600000004	92.12865699999975
19	88.1873370000003	90.25178400000004
20	88.30919499999982	89.39762199999996
21	88.04772000000003	89.78546000000006
22	89.15494799999988	89.91057699999965
23	88.52565299999969	90.00702000000001
24	88.82988399999977	89.72220900000002
25	87.89214100000027	90.92437500000005
26	87.81264800000008	88.91824000000042
27	88.26395100000036	90.71504999999979
28	88.11187400000017	88.81896299999971
29	88.8326049999996	91.90608699999984
30	88.0618129999998	88.51013399999965
Total	2657.511897999999	2707.5782929999987
Rata-Rata	88.5837299333333	90.252609766666633



Gambar 6.9 Grafik Hasil Pengujian Waktu Autentikasi Berdasarkan Algoritme

Tabel 6.21, Tabel 6.22, dan Gambar 6.9 merupakan hasil pengujian autentikasi waktu yang dibutuhkan masing masing algoritme. Dari Tabel 6.21 dan Tabel 6.22 juga menunjukkan rata-rata waktu yang dubutuhkan masing-masing algoritme. Pada hasil akhir ditemukan selisih waktu sebesar 50.0663949 ms dan selisih rata rata waktu sebesar 1.66887983 ms maka dapat disimpulkan bahwa algoritme BLAKE2S lebih cepat 0.981%dibandingkan dengan algoritme HMAC-SHA256.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil penelitian dan analisis yang telah dilakukan dalam penelitian, dapat diambil kesimpulan beberapa hal mengenai autentikasi REST API dengan JSON Web Token dengan menggunakan algoritme BLAKE2S sebagai berikut

1. Implementasi algoritme BLAKE2S pada *library* JSON Web Token berhasil dilakukan dengan menambahkan pilihan untuk menggunakan algoritme BLAKE2S pada JSON Web Signature (JWS) dan algoritme BLAKE2S disertakan pada konfigurasi JSON Web Algorithm (JWA).
2. Algoritme BLAKE2S memiliki waktu rata-rata autentikasi 88.5837299333333 ms sedangkan algoritme HMAC-SHA256 memiliki waktu rata-rata autentikasi 90.252609766666633 ms dengan demikian algoritme BLAKE2S lebih cepat dalam melakukan autentikasi sebesar 0.981 Persen. Hal ini memberikan alternatif algoritme *hash* pada JSON Web Token.

7.2 Saran

Pada arsitektur *client-server* REST, JSON Web Token memberikan nilai autentikasi dan nilai integritas dari algoritme *hash* yang digunakan. Dalam implementasi nyata dari arsitektur *client-server* REST dibutuhkan nilai *confidentiality* untuk memberikan kerahasiaan pada pesan yang dikirim juga kerahasiaan token yang digunakan maka saran yang dapat penulis berikan untuk penelitian selajuntnya yaitu penggunaan algoritme enkripsi pada protokol HTTPS. Protokol HTTPS memberikan keamanan berupa kerahasiaan pesan yang dikirim agar pesan yang dikirim melalui jaringan internet tidak bisa dibaca oleh pihak yang tidak terlibat dalam komunikasi. Algoritme yang tersedia untuk protokol HTTPS diantaranya algoritme RSA atau kombinasi algoritme RSA dengan algoritme SHA256. Penelitian ini diharapkan dapat digunakan sebagai rujukan untuk penelitian maupun pengembangan selanjutnya.

DAFTAR PUSTAKA

Aumasson, Jean-Philipper., 2013. *BLAKE2: Simpler, Smaller, Fast as MD5*. Tersedia di: <https://blake2.net/blake2.pdf>

Bellare, Mihir. Canetti, Ran. Krawczyk, Hugo. 1996. Keying Hash Function for Message Authentication. Tersedia di: < http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.8430 >

Bernstein, Daniel J., 2008. *ChaCha, a Variant of Salsa20*. The University of Illinois, Chicago. Tersedia di : < https://cr.yp.to/chacha/chacha-20080128.pdf>

Bradley, john. Nat, Sakimura., 2015. *JSON Web Token*. Internet Engineering Task Force. Tersedia di: < https://tools.ietf.org/html/rfc7519>

Dunkelman, Orr., 2007. *A framework for iterative hash function –HAIFA*. Cryptology ePrint Archieve, Report 2007/278. Tersedia di: <http://eprint.iacr.org/2007/278>.

Fielding, Roy T., 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation. University of California, Irvine.

Fielding, Roy T. Richard N., 2005. *Principled Design of the Modern Web Archictecture*. ACM Transactions on Internet Technology (TOIT).

Hair, J.F., Black, W.C., Babin. B. J. & Andreson R. E., 2010. *Multivariate Data Analysis*. Edisi 7. Upper Saddle River: Prentice Hall

JSON Organization., 1999. *Introducing JSON*. JSON Organization. Tersedia di: < https://www.json.org/>

Khovratovich, Dmitry. Rechberger, Christian., 2011. *Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family*. IACR Cryptology ePrint Archive. 2011:286.

Lamberger, Mario. Mendel, Florian., 2011. *Higher-Order Differential Attack on Reduced SHA-256*. IACR Cryptology ePrint Archive. 2011:37

Munir, Rinaldi., 2004. *Digital Signature Standard (DSS)*. Bandung: Institue Teknologi Bandung.

Munir, Rinaldi., 2004. *Fungsi Hash Satu-Arah dan Algoritma MD5*. Bandung: Institute Teknologi Bandung. Tersedia di: < http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/Fungsi%20Hash%20dan%20Algoritma%20MD5.pdf> [Diakses 25 Juli 2018]

Wang, Xiaoyun. Yin, Yiqun Lisa., 2005. *Finding Collisions in the Full SHA-1*. Advances in Cryptology – CRYPTO 2005. Springer, Berlin, Heidelberg: 17–36.

World Wide Web Consortium., 2004. *Relationship to the World Wide Web and REST Architectures*. Tersedia di: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest> [Diakses 7 Februari 2018]