Keamanan RESTful *Web Service* Menggunakan *JSON Web Token* (JWT) HMAC SHA-512

Alam Rahmatulloh[1]*, Heni Sulastri[2], Rizal Nugroho[3]

***Abstract*—Day to day information technology is constantly evolving, allowing a wide range of technologies, programming languages, and diverse architectures to keep popping up. It makes a new problem because at present all these differences must still be able to generate an interconnected information. It needs system integration. Currently, Web Service (WS) is a solution in system integration because it can be used without looking at the platform, architecture, or programming language used by different sources. But, on WS, the existing security is still considered less. Implementation of JSON Web Token (JWT) on WS is very influential in data security. JWT is an authentication mechanism on WS, but the application of standard JWT with HMAC SHA-256 algorithm is still not optimal. Therfore, this study discussed JWT security optimization with HMAC SHA-512 algorithm, which according to some researches, this algorithm will be better than SHA-256 if compiled on 64-bit architecture. The result of this research is that the use of SHA-512 produces a better time of 1% than SHA-256, but in SHA-512 token size is 2% larger than SHA-256.**

***Intisari*—Teknologi informasi dari hari ke hari terus berkembang, sehingga berbagai macam teknologi, bahasa pemrograman, dan arsitektur beragam terus bermunculan. Hal tersebut menjadikan permasalahan baru karena pada zaman sekarang semua perbedaan tersebut harus tetap bisa menghasilkan sebuah informasi yang saling terhubung. Maka, diperlukan integrasi sistem. Saat ini *Web Service* (WS) adalah solusi dalam integrasi sistem karena tanpa melihat *platform*, arsitektur maupun bahasa pemrograman yang digunakan oleh sumber berbeda. Namun, pada WS keamanan yang ada masih dirasa kurang. Penerapan *JSON Web Token* (JWT) pada WS sangat berpengaruh dalam hal keamanan data. JWT merupakan mekanisme autentikasi pada WS, tetapi penerapan JWT standar dengan algoritme HMAC SHA-256 masih belum optimal, sehingga pada makalah ini dibahas optimasi keamanan JWT dengan algoritme HMAC SHA-512, yang menurut beberapa penelitian algoritme ini akan lebih baik dibandingkan SHA-256 jika dikompilasi pada arsitektur 64-bit. Hasil menunjukkan, penggunaan SHA-512 menghasilkan waktu yang lebih baik 1% dibandingkan SHA-256. Namun pada segi ukuran *token* yang dihasilkan, SHA-512 lebih besar 2% dibandingkan SHA-256.**

***Kata Kunci*— HMAC, *JSON Web Token*, RESTful, SHA-512, *Web Service*.**

I. PENDAHULUAN

Teknologi informasi terus berkembang, memberikan pengaruh besar terhadap organisasi maupun individu. Perkembangan teknologi bertujuan untuk memenuhi kebutuhan bagi pengguna. Komputasi terdistribusi adalah salah satu teknologi informasi yang dapat melakukan komputasi pada banyak mesin dan dimanfaatkan banyak mesin. Komputasi terdistribusi ditemukan setelah adanya teknologi web. Maksud dari teknologi ini adalah *Web Service* (WS). Dalam perkembangan bisnis, WS sangat diperlukan dalam integrasi sistem karena tanpa melihat *platform*, arsitektur maupun bahasa pemrograman yang digunakan oleh sumber berbeda. Keamanan WS berada ke dalam sepuluh kerentanan teratas dalam keamamanan *Application Programming Interface* (API) *Web Service* yang kurang terlindungi menurut *The Open Web Application Security Project* (OWASP) [1].

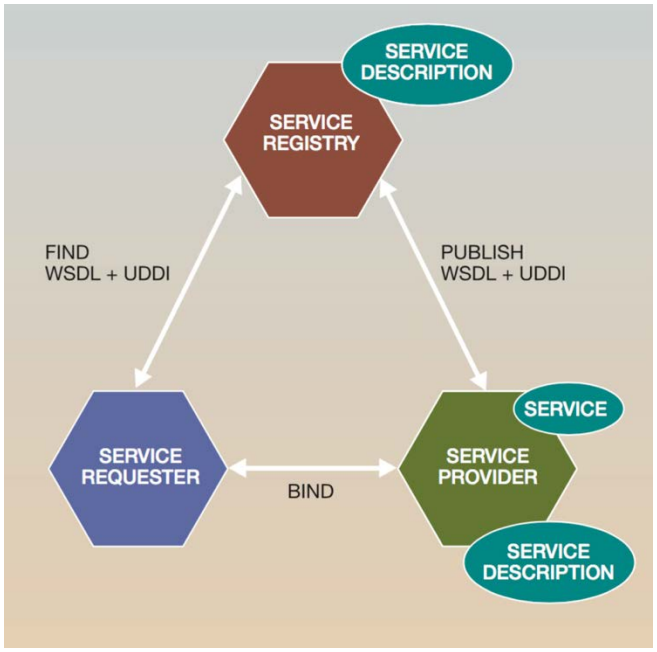
Perkembangan WS saat ini yang sedang tren yaitu *REpresentational State Transfer* (REST) dan *Simple Object Access Protocol* (SOAP). Hasil penelitian yang telah dilakukan pada aplikasi *mobile computing* menunjukkan bahwa ukuran pesan pada RESTful *Web Service* mencapai sembilan sampai sepuluh kali lebih kecil dibandingkan ukuran pesan dari WS berbasis SOAP [2]. Namun, REST sangat rendah dalam segi keamanan [3]. Mengamankan RESTful WS mencakup mengamankan data serta seluruh komunikasi untuk melindungi kerahasiaan dan integrasi data [4]. Untuk mengatasi masalah tersebut, digunakan *JSON Web Token* (JWT). Langkah ini telah dilakukan pada penelitian sebelumnya, yang menggunakan JWT dengan algoritme HMAC SHA-256 yang masih umum digunakan, sehingga dapat menjadi ancaman tersendiri bagi keamanan RESTful WS [5]. Hasil sebuah penelitian menyatakan, perbandingan penerapan algoritme SHA-256 dan SHA-512 pada arsitektur intel 64-bit mengasilkan kinerja SHA-512 50% lebih baik dibandingkan dengan SHA-256 [6]. Kemudian diperkuat oleh penelitian lain bahwa algoritme SHA-512 memiliki kinerja yang jauh lebih baik daripada algoritme SHA-256 jika dikompilasi terhadap arsitektur 64-bit dan dijalankan pada mesin 64-bit [7]. Oleh karena itu, makalah ini membahas mengenai kecepatan dan ukuran data pada keamanan RESTful WS menggunakan JWT dengan algoritme HMAC SHA-512 pada arsitektur 64-bit.

II. KEAMANAN *WEB SERVICE*

A. Web Service (WS)

WS merupakan sebuah perangkat lunak yang tidak terpengaruh oleh *platform*, arsitektur, maupun bahasa pemrograman, yang menyediakan layanan atau *method-method* untuk pertukaran data yang dapat diakses oleh *network* [8]. Contoh implementasi dari WS antara lain adalah SOAP dan REST. Arsitektur WS ditunjukkan pada Gbr. 1.

[1,2,3]*Program Studi Teknik Informatika, Universitas Siliwangi Tasikmalaya, Jalan Siliwangi No.24 Kota Tasikmalaya, Tasikmalaya, 46115, INDONESIA* (*tlp:* (*0265*) *323537; fax:* (*0265*) *325812; e-mail: ft@unsil.ac.id*).
(*) *Corresponding author*



Gbr. 1 Arsitektur *Web Service* [9].

B. *RESTful*

Konsep REST pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000 [10]. REST merupakan standar arsitektur komunikasi berbasis web yang selalu digunakan terhadap pengembangan layanan berbasis web. Pada umumnya, *Hypertext Transfer Protocol* (HTTP) berperan sebagai protokol untuk melakukan komunikasi data [11]. Sistem yang menggunakan prinsip-prinsip dari REST dapat disebut dengan "RESTful" [12]. Penetapan indentifikasi terhadap *resource* dilakukan oleh *Universal Resource Identifiers* (URIs) atau *global ID*. *Resource* diperkenalkan dengan format teks, JSON, atau XML. Pada umumnya, format yang digunakan adalah JSON dan XML.

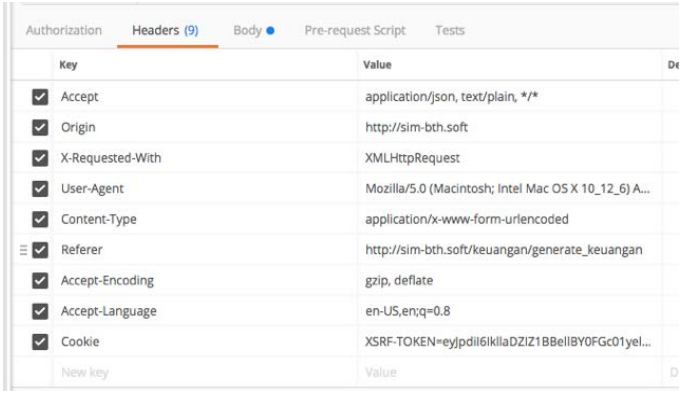
Cara kerja RESTful WS yaitu bermula dari *client* mengirimkan sebuah data atau *request* melalui *HTTP Request*, kemudian *server* merespons melalui *HTTP Response*. Komponen dari *HTTP Request* adalah sebagai berikut.

- *Verb*. *HTTP Method* yang digunakan di antaranya GET (hanya menyediakan akses baca pada *resource*), PUT (digunakan untuk menciptakan *resource* baru), DELETE (digunakan untuk menghapus *resource*), POST (digunakan untuk memperbarui *resource* yang ada atau membuat *resource* baru), OPTIONS (digunakan untuk mendapatkan operasi yang didukung pada *resource*).
- *Uniform Resource Identifier* (URI) untuk mengidentifikasikan lokasi *resource* pada *server*.
- *HTTP Version*, menjelaskan versi dari HTTP yang akan digunakan, contohnya HTTP v1.1.
- *Request Header*, berisi metadata untuk *HTTP Request*. Contohnya adalah tipe *client*/*browser*, format yang didukung oleh *client*, format dari *body* pesan, dan *setting cache*.
- *Request Body*, yaitu konten dari data.

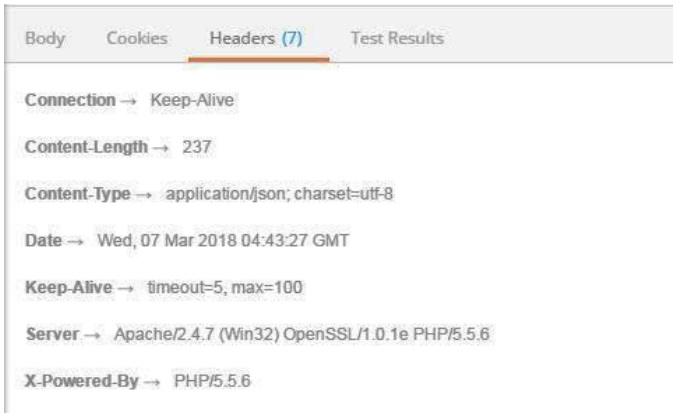
Sedangkan komponen dari *HTTP Response* adalah sebagai berikut.

- *Status/Response Code*, menjelaskan status *server* pada *resource* yang di-*request*. Contohnya 404, artinya *resource* tidak ada dan 200 *response OK*.
- *HTTP Version*, menunjukkan versi dari HTTP yang digunakan. Contohnya HTTP v1.1.
- *Response Header*, berisi metadata untuk *HTTP Response*. Contohnya tipe *server*, panjang *content*, tipe *content*, dan waktu *response*.
- *Response Body*, yakni konten dari data yang diberikan.

Ada dua bagian pesan yang digunakan untuk membangun komunikasi dengan *server*, yaitu pesan *Header* dan pesan *Body*. *HTTP header* adalah "catatan" kecil pada setiap transaksi data HTTP yang dikirim *browser*/*server* baik pada proses *request* maupun *response*. Ada beberapa elemen yang terdapat pada *HTTP header*, contohnya *HTTP status*, *cache-control*, jenis *Web server*, dan sebagainya. *Request Headers* ditunjukkan pada Gbr. 2 dan *Response Headers* ditampilkan pada Gbr. 3.

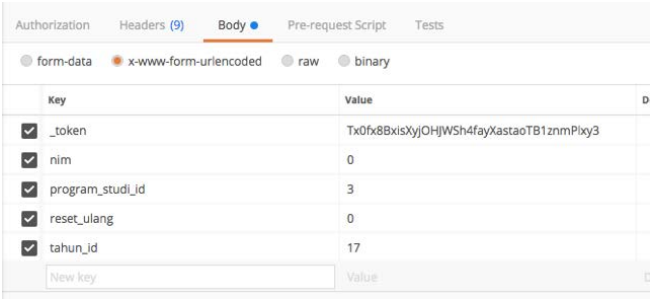


Gbr. 2 Contoh *Request Headers*.



Gbr. 3 Contoh *Response Headers*.

Bagian *body* dari HTTP berisi data yang akan dikirimkan. Data yang dikirimkan dapat berisi apapun, misalnya HTML (paling umum), gambar, data biner apapun, atau boleh tidak berisi (kosong). Contoh *body request* ditunjukkan pada Gbr. 4 dan contoh *body response* pada Gbr. 5.



Gbr. 4 Contoh *Request Body*.

Gbr. 5 Contoh *Response Body*.

C. *JWT (JSON Web Token)*

JWT ini adalah sebuah *token* berbentuk *string* JSON yang sangat padat (ukurannya), informasi mandiri yang gunanya sendiri untuk melakukan sistem autentikasi dan pertukaran informasi. Karena bentuknya kecil, *token* JWT dapat dikirim melalui URL, parameter HTTP POST atau di dalam *Header* HTTP, dan juga karena ukurannya yang kecil maka dapat ditransmisikan dengan lebih cepat. Disebut informasi mandiri karena isi dari *token* yang dihasilkan memiliki informasi dari pengguna yang dibutuhkan, sehingga tidak perlu *query* ke basis data lebih dari satu kali. *Token* tersebut dapat diverifikasi dan dipercaya karena sudah di-*sign* secara digital. *Token* JWT dapat di-*sign* dengan menggunakan *secret* (algoritme HMAC) atau pasangan *public*/*private key* (algoritme RSA). Proses *login* yang dilakukan tidak seperti aplikasi *website* biasa, tetapi menggunakan *session* untuk mengingat yang sedang melakukan proses *login*. Namun, API hanya menggunakan konsep JWT yang dapat disebut "jot" [13]. JWT tidak bergantung pada bahasa program tertentu. Struktur JWT terdiri atas tiga bagian yang dipisahkan oleh titik (“.”), yaitu *header*, *payload*, dan *signature*, seperti ditunjukkan pada Gbr. 6.

```
xxxxxx.yyyyyyy.zzzzzzzzz
header.payload.signature
```

Gbr. 6 Struktur *JSON Web Token*.

Header biasanya terdiri atas dua bagian, yaitu tipe *token*, yakni JWT, dan algoritme *hashing* yang digunakan, seperti HMAC SHA-256 atau RSA dan lainnya. Contoh *header* ditunjukkan pada Gbr. 7.

```
{
  “alg” : “HS256”,
  “typ” : “JWT”,
}
```

Gbr. 7 JWT *Header*.

Payload, bagian kedua, berisi *klaim*. *Klaim* adalah pernyataan tentang suatu entitas (biasanya pengguna) dan *metadata* tambahan. Ada tiga jenis *klaim*, yaitu *reserved*, *public*, dan *private claims*. Bagian kedua (*payload*) diperlihatkan pada Gbr. 8.

```
{
  “sub”: ”123456789”,
  “name”: “jhon Doe”,
  “admin”: true
}
```

Gbr. 8 *JWT Payload*.

Bagian ketiga dari JWT adalah *signature*, berisi *hash* dari komponen-komponen *header*, *payload*, dan kunci rahasia. Contoh *JWT Signature* ini menggunakan algoritme HMAC SHA-256. *Signature* dapat dibuat dengan cara seperti pada Gbr. 9.

```
HMACSHA256(
   base64UrlEncode(header)+”.”+
   base64UrlEncode(payload),
   secret)
```

Gbr. 9 *JWT Signature*.

Keluarannya adalah tiga *string Base64* yang dipisahkan oleh titik-titik yang dapat dengan mudah dilewatkan dalam HTML dan HTTP. Apabila isi *Header* atau *Payload* diubah, maka isi *Signature* menjadi tidak *valid* [14]. *Signature* dibentuk dengan menggunakan *header* dan *payload* sehingga JWT mampu memberikan kemudahan bagi *client* untuk mengakses sumber daya tanpa harus *login* berulang memasukkan *username* dan *password*. *Token* dapat dipanggil melalui AJAX ke *server* karena panggilan dapat menggunakan *HTTP header* untuk mengirimkan informasi penggunanya.

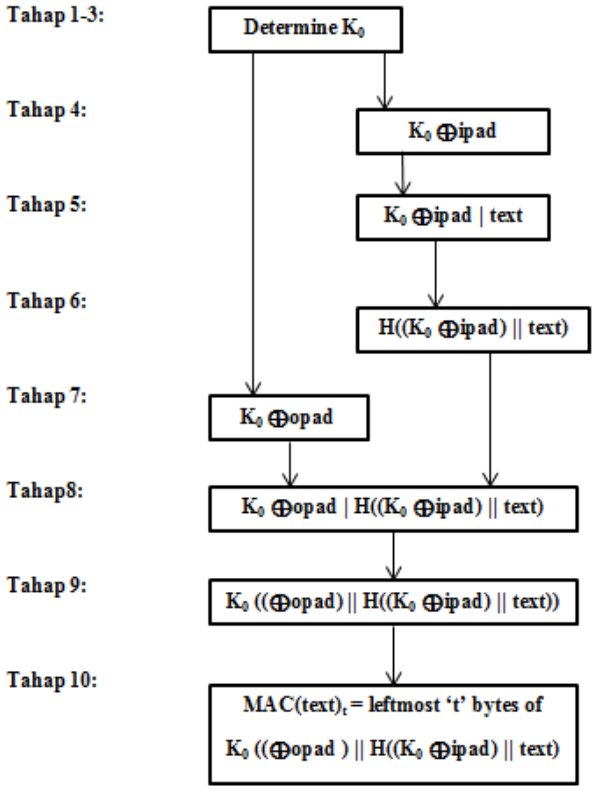
D. *Keyed-Hash Massage Authentication Code* (*HMAC*)

HMAC adalah teknik autentikasi pesan dengan memanfaatkan fungsi *hash* terhadap pesan dan kemudian mengenkripsi pesan tersebut dengan menggunakan kunci *private*. HMAC dibuat oleh Mihir Bellare, Ran Canetti, dan Hugo Krawczyk pada tahun 1996. Algoritme HMAC dapat dijabarkan menjadi sepuluh langkah, seperti diperlihatkan pada Gbr. 10.

Penjelasan Gbr. 10 diagram algoritme HMAC adalah :

- Tahap 1: Jika panjang K = B, atur K_0 = K, kemudian ke tahap 4.
- Tahap 2: Jika panjang K > B, *hash* K untuk memperoleh L *string byte*, kemudian *append* dengan (B-L) angka 0 untuk menghasilkan *string byte* K_0 yang memiliki panjang sama dengan B. Kemudian ke tahap 4.
- Tahap 3: Jika panjang K < B, *append* angka 0 sebanyak (B-K) untuk menghasilkan *string byte* K_0 yang memiliki panjang sama dengan B. Kemudian ke tahap 4.
- Tahap 4: Melakukan *XOR* antara K_0 dengan *ipad* untuk menghasilkan *string byte* memiliki panjang yang sama seperti B.

- Tahap 5: *Append string 'text'* terhadap hasil *string* dari tahap 4 tadi.
- Tahap 6: Melakukan H untuk *string* yang dihasilkan oleh tahap 5.
- Tahap 7: Melakukan *XOR* antara K_0 dengan *opad*.
- Tahap 8: Melakukan *append string* dari tahap 6 ke dalam *string* dari tahap *7*.
- Tahap 9: Melakukan H untuk *string* yang dihasilkan dari tahap 8.
- Tahap 10: Mengambil *leftmost byte* sebanyak t dari *string* yang dihasilkan tahap 9.



Gbr. 10 Diagram algoritme HMAC [15].

Penjelasan dari *variabel* yang digunakan di dalam diagram dan algoritme pada Gbr. 10 ditunjukkan pada Tabel I [16] .

Fungsi yang dapat diperoleh dalam algoritme HMAC adalah sebagai berikut.

- Untuk memakai fungsi *hash*, tanpa melakukan perubahan pada fungsi *hash* karena telah tersedia dan mudah didapatkan.
- Untuk mengatur kunci *private* dengan mudah dan mempertahankan kemanan.
- Untuk mendapatkan pemahaman yang lebih dalam terhadap analisis kriptografi mengenai kekuatan mekanisme autentikasi yang terdapat dalam fungsi *hash*.
- Untuk mempermudah melakukan perubahan atau mengganti fungsi *hash* yang digunakan jika ada algoritme *hash* baru yang memiliki kinerja lebih cepat atau memiliki keamanan lebih baik.

TABEL I
DIAGRAM ALGORITME HMAC

Variabel	Deskripsi
B	Ukuran *block* (dalam *byte*) dari masukan ke fungsi *Hash*
H	Fungsi *Hash* yang digunakan
Ipad	*inner pad* (0x3636...36)
K	Kunci *private* yang diketahui oleh pengirim dan penerima
K_0	Kunci yang telah diproses seperlunya agar panjangnya B
L	Ukuran *block* (dalam *byte*) dari keluaran fungsi *Hash*
Opad	*outer pad* (0x5c5c...5c)
T	Jumlah *byte* dari *MAC* yang diinginkan
Text	Pesan/informasi yang akan dimanipulasi

E. SHA (Secure Hash Algorithm)

SHA adalah fungsi *hash* satu arah *(one-way hash function)* yang dibuat oleh *National Institute of Standards and Technology* (NIST) dan digunakan bersama *Digital Signature Standard* (DSS) [17]. Persamaan dari fungsi *hash* yaitu sebagai berikut.

$$h = H(M) \quad (1)$$

Sifat-sifat *hash* adalah sebagai berikut [18].

- Fungsi H dapat digunakan pada blok data yang memiliki kapasitas data berapa saja.
- Fungsi H menghasilkan nilai (h) dengan memiliki panjang yang tetap (*fixed-length output*).
- H(x) dapat diperoleh secara mudah dari setiap nilai x yang diberikan.
- Setiap hasil nilai h tidak mungkin dapat melakukan pengembalian nilai x sehingga H(x) = h. Oleh sebab itu, fungsi H disebut fungsi *hash* satu-arah (*one-way hash function*).
- Untuk setiap nilai x yang dimasukkan, tidak dapat melakukan pencarian $y \neq x$ sehingga H(y) = H(x).
- Tidak dapat mencari pasangan x dan y sehingga H(x) = H(y).

III. METODOLOGI

Metodologi yang digunakan dalam penyusunan makalah ini adalah sebagai berikut.

A. Studi Literatur

Tahapan pengumpulan data ini dilakukan agar penelitian lebih fokus dan tidak terjadi penelitian yang serupa.

B. Analisis Kebutuhan Perangkat Lunak

Analisis kebutuhan perangkat lunak dilakukan untuk menggali kebutuhan perangkat lunak yang dibangun.

C. Implementasi Perangkat Lunak

Implementasi perangkat lunak dilakukan dengan membuat aplikasi WS sederhana dengan menerapkan algoritme HMAC SHA-256 dan SHA-512.

D. Pengujian dan Perbandingan

Pengujian dan perbandingan kinerja pada WS dilakukan dengan penerapan JWT menggunakan HMAC SHA-256 dan HMAC SHA-512.

IV. HASIL DAN PEMBAHASAN

A. Studi Literatur

Telah dilakukan pengumpulan data dan studi literatur pada penelitian-penelitian sebelumnya yang berhubungan dengan topik dan menjadi dasar dalam makalah ini. Di antaranya adalah alasan pembahasan keamanan pada RESTful WS yang dibahas oleh OWASP dan Snehal, kemudian sudah dilakukan alternatif keamanan pada WS menggunakan JWT SHA-256 oleh Penidas, tetapi dirasa masih memiliki kelemahan. Penelitian yang dilakukan Shay dan Amudi menghasilkan perbandingan kinerja SHA-512 lebih baik 50% dibandingkan SHA-256 pada arsitektur 64-bit.

B. Analisis Kebutuhan Perangkat Lunak

Berdasarkan literatur yang telah diperoleh, maka pada makalah ini spesifikasi sistem menggunakan:

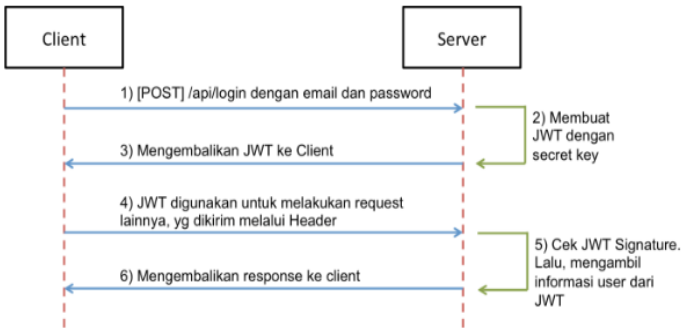
- *processor AMD A8-4500M APU with Radeon (tm) HD Graphic 1.90Ghz,*
- *memory (RAM) 8,00GB,*
- *VGA Card AMD Radeon HD 7640G,*
- *system type 64bit Operating System,*
- *harddisk 500GB,* dan
- *tools Postman RST Client.*

C. Implementasi Perangkat Lunak

Untuk membuktikan bahwa kinerja algoritme HMAC SHA-512 lebih optimal dibandingkan SHA-256, maka dibuatlah perangkat lunak WS sederhana berbasis web untuk penerapan dan perbandingan terhadap JWT menggunakan HMAC SHA-256 dan SHA-512. Tujuan utama dari JWT adalah pengamanan data antar sistem yang melakukan pertukaran data dengan melakukan *request* oleh *client* yang harus menyertakan *token*. Secara umum, cara kerja JWT diperlihatkan pada Gbr. 11.

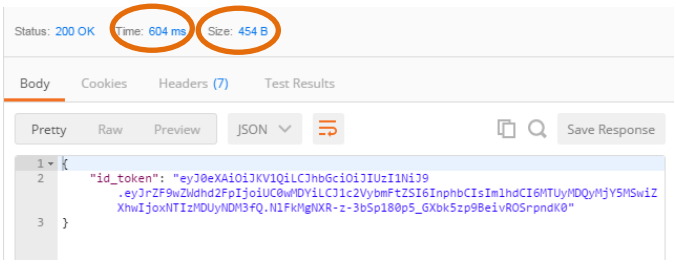
D. Pengujian dan Perbandingan

Pengujian dilakukan dengan *Postman* yang memiliki fungsi sebagai *REST Client* atau aplikasi yang digunakan untuk melakukan uji coba REST API yang telah dibuat. Ada dua proses yang dilakukan pada *Postman* yaitu POST dan GET. POST untuk mengirimkan parameter yang berupa *username* dan *password* yang valid untuk menghasilkan *token*. *Token* ini yang berfungsi sebagai kunci agar mendapat akses untuk melakukan *request* selanjutnya. Proses GET dilakukan dengan memasukkan kunci atau *token* yang telah diperoleh pada proses POST untuk mendapatkan data yang diinginkan dengan melakukan *request* ke */api/restdata/*GET*Token*. Ketika melakukan *request*, pada umumnya *token* dikirimkan pada *HTTP header*. *Default key* untuk membawa *token* adalah "*Authorization*". Biasanya secara *defaut token* akan selalu diperbarui setelah *HTTP request* dilakukan, dan dapat diakses pada *response header*. Tampilan hasil penerapan dari JWT SHA-256 dan SHA-512 ditunjukkan pada Gbr. 12 sampai Gbr. 15.

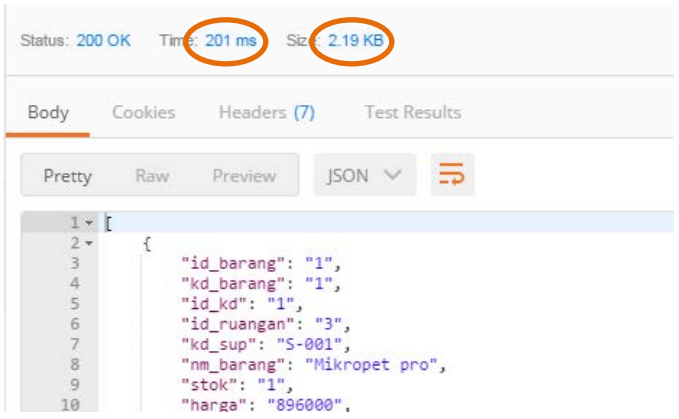


Gbr. 11 Cara kerja JWT.

1) SHA-256: Gbr. 12 menunjukan proses *parsing* data menggunakan SHA-256 yang dihasilkan proses POST dengan waktu 604 ms dan ukuran 454 *byte*, sedangkan Gbr. 13 adalah hasil proses GET SHA-256 yang mendapatkan waktu 201 ms dengan ukuran 2,19 kb.

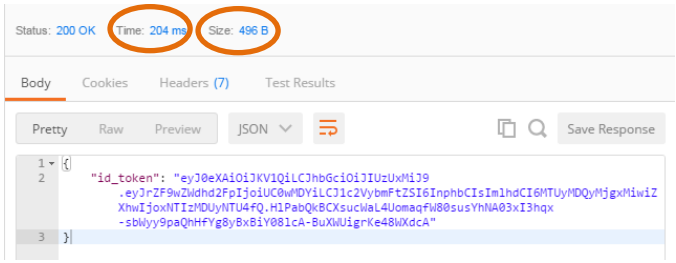


Gbr. 12 Proses POST SHA-256.

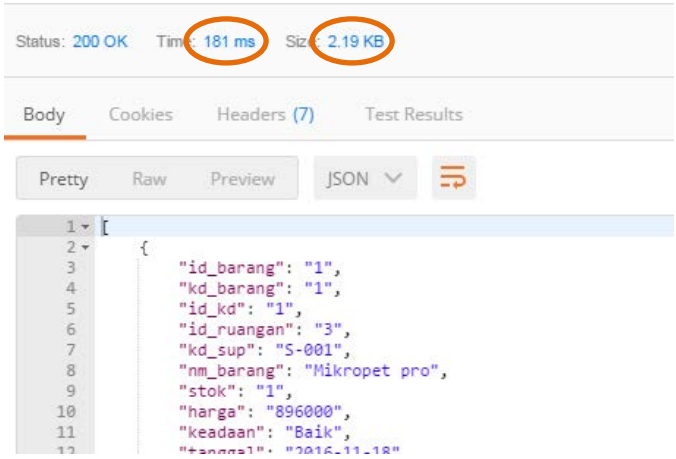


Gbr. 13 Proses GET SHA-256.

2) SHA-512: Gbr. 14 menunjukan proses *parsing* data menggunakan SHA-512 yang didapatkan proses POST dengan waktu 204 ms dan ukuran 496 *byte*, sedangkan Gbr. 15 menunjukkan hasil proses GET SHA-512 yang mendapatkan waktu 181 ms dengan ukuran 2,19 kb. Percobaan proses GET pada SHA-256 dan SHA-512 mengunakan aplikasi yang sama, sehingga ukuran data yang dihasilkan sama dan hanya memiliki perbedaan pada segi kecepatan.



Gbr. 14 Proses POST SHA-512.



Gbr. 15 Proses GET SHA-512.

Selanjutnya, dibahas perbandingan antara Token JWT SHA-256 dan SHA-512. Perbandingan yang terlihat terdapat pada *token* yang dihasilkan. *Token* yang dihasilkan oleh SHA-256 yaitu berukuran 256 *bit* sebagai berikut.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJrZF9wZWdhd2FpIjoiUC0wMDYiLCJ1c2VybmFtZSI6InphbCIsImlhdCI6MTUyMDQyMjY5MSwiZXhwIjoxNTIzMDUyNDM3fQ.NlFkMgNXR-z-3bSp180p5_GXbk5zp9BeivROSrpndK0
```

Sementara itu, *token* yang dihasilkan oleh SHA-512 yaitu berukuran 512 *bit*:

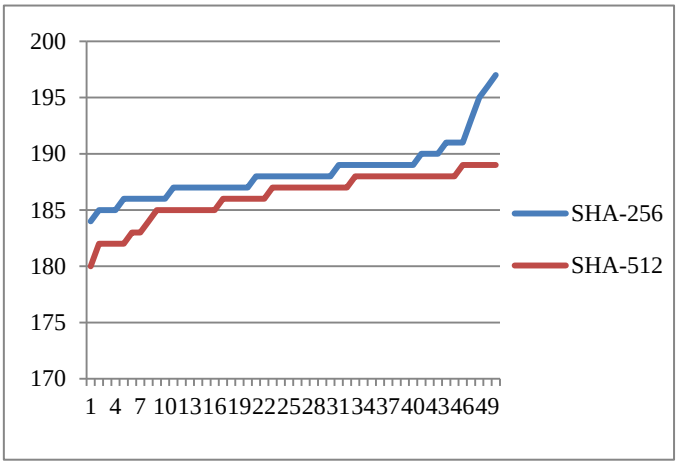
```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJrZF9wZWdhd2FpIjoiUC0wMDYiLCJ1c2VybmFtZSI6InphbCIsImlhdCI6MTUyMDQyMjgxMiwiZXhwIjoxNTIzMDUyNTU4fQ.HlPabQkBCXsucWaL4UomaqfW80susYhNA03xI3hqx-sbWyy9paQhHfYg8yBxBiY08lcA-BuXWUigrKe48WXdcA
```

Dari kedua *token* tersebut, dapat dilihat bahwa panjang *token* sangatlah berbeda. SHA-512 memiliki *token* lebih panjang dibandingkan SHA-256 dikarenakan perbedaan bit. Untuk melihat perbandingan kinerja pada kedua algoritme tersebut, dilakukan pengujian dari sisi kecepatan dan ukuran *token* yang dihasilkan. Perbandingan diperlihatkan pada Tabel II.

Tabel II adalah proses pengujian perbandingan kinerja dua puluh kali dari lima puluh kali percobaan terhadap SHA-256 dan SHA-512 dari segi kecepatan (ms) dan ukuran data *token*. Dari percobaan tersebut, dapat dilihat juga grafik lima puluh percobaan perbandingan waktu yang ditunjukkan pada Gbr. 16.

TABEL II
PERBANDINGAN ALGORITME HMAC SHA-256 DAN SHA-512

No.	Kecepatan (m/s)		Ukuran (kb)	
	SHA-256	SHA-512	SHA-256	SHA-512
1	184	180	2,28	2,32
2	185	182	2,28	2,32
3	185	182	2,28	2,32
4	185	182	2,28	2,32
5	186	182	2,28	2,32
6	186	183	2,28	2,32
7	186	183	2,28	2,32
8	186	184	2,28	2,32
9	186	185	2,28	2,32
10	186	185	2,28	2,32
11	187	185	2,28	2,32
12	187	185	2,28	2,32
13	187	185	2,28	2,32
14	187	185	2,28	2,32
15	187	185	2,28	2,32
16	187	185	2,28	2,32
17	187	186	2,28	2,32
18	187	186	2,28	2,32
19	187	186	2,28	2,32
20	187	186	2,28	2,32
Rata-rata	**186,25**	**184,1**	**2,28**	**2,32**



Gbr. 16 Grafik perbandingan SHA-256 dan SHA-512.

SHA-256 memiliki rata-rata kecepatan 188,38 ms sedangkan SHA-512 memliki rata-rata kecepatan 186,26 ms. Dapat dilihat bahwa kecepatan SHA-256 lebih lambat jika dibandingkan SHA-512. Dalam lima puluh percobaan tersebut, nilai minimum yang diperoleh adalah 180 ms oleh SHA-512 dan nilai maksimum yang diperoleh adalah 189 ms, sedangkan nilai minumum yang diperoleh SHA-256 adalah 184 ms dan nilai maksimum adalah 195 ms. Pada segi ukuran *token* SHA-256 rata-rata yaitu 2,28 kb, sedangkan ukuran *token* SHA-512 rata-rata adalah 2,32 kb. Perbedaan tersebut dikarenakan nilai bit yang berbeda, sehingga menjadikan keamanan pertukaran data lebih baik karena *token* yang

dihasilkan SHA-512 lebih panjang jika dibandingkan dengan *token* SHA-256.

V. KESIMPULAN

Kesimpulan yang dapat diambil yaitu hasil penerapan algoritme HMAC SHA-512 pada JWT dalam WS dan pada arsitektur 64-bit menghasilkan kinerja yang lebih baik. SHA-512 lebih cepat 1% dibandingkan dengan SHA-256. Hal tersebut dapat dilihat dari hasil percobaan, tetapi perbandingan penerapan algoritme SHA-256 dan SHA-512 tidak mencapai 50% jika diterapkan pada JWT. Hasil dari percobaan yang telah dilakukan membuktikan bahwa arsitektur 64-bit memiliki kinerja lebih baik dalam penerapan algoritme HMAC pada JWT dalam WS. Namun, jika dilihat dari segi ukuran data *token*, SHA-512 menghasilkan nilai *hash* 2% lebih besar daripada SHA-256. Hal ini tentunya menjadikan keamanan pertukaran data lebih baik karena *token* lebih panjang.

REFERENSI

[1] (2017) "OWASP Top 10 – 2017 The Ten Most Critical Web Application Security Risks," [Online], https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf, tanggal akses: 12-Des- 2017.

[2] S. Mumbaikar dan P. Padiya, "Web Services Based on SOAP and REST Principles," *Int. J. Sci. Res. Publ.*, Vol. 3, No. 5, hal. 3–6, 2013.

[3] V. Kumari, "Web Services Protocol : SOAP vs REST," *IJARCET*, Vol. 4, No. 5, hal. 2467–2469, 2015.

[4] K. V. Kanmani dan P. S. Smitha, "Survey on Restful Web Services Using Open Authorization (Oauth)," *IOSR J. Comput. Eng.*, Vol. 15, No. 4, hal. 53–56, 2013.

[5] P. F. Tanaem, D. Manongga, dan A. Iriani, "RESTFul Web Service untuk Sistem Pencatatan Transaksi Studi Kasus PT. XYZ," *Jurnal Teknik Informatika dan Sistem Informasi*, Vol. 2, No. 1, hal. 1–10, 2016.

[6] S. Gueron, S. Johnson, dan J. Walker, "Sha-512/256," *Proc. 2011 Eighth Int. Conf. Inf. Technol. New Gener. (ITNG '11)*, 2011, hal. 354–358.

[7] A. Sebastian, "Implementasi dan Perbandingan Performa Algoritma Hash SHA-1, SHA-256, dan SHA-512," Skripsi, Institut Teknologi Bandung, Bandung, Indonesia, 2007.

[8] A. Gustavo, F. Casati, H. Kuno, dan M. Vijay, *WEB SERVICES*, New York, USA: Springer-Verlag, 2004.

[9] K. D. Gottschalk, S. Graham, H. Kreger, dan J. Snell, "Introduction to Web Services Architecture," *IBM Syst. J.*, Vol. 41, No. 2, hal. 170–177, 2002.

[10] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," *Building*, Vol. 54, hal. 162, 2000.

[11] L. Richardson dan S. Ruby, *RESTful Web Services*, O'Reilly Media, 2007.

[12] C. J. Su dan C. Y. Chiang, "Enabling Successful Collaboration 2.0: A REST-based Web Service and Web 2.0 Technology Oriented Information Platform for Collaborative Product Development," *Comput. Ind.*, Vol. 63, No. 9, hal. 948–959, 2012.

[13] (2017) "JSON Web Tokens - jwt.io," [Online], https://jwt.io/, tanggal akses: 12-Dec-2017.

[14] M. Jones, J. Bradley, dan N. Sakimura, (2017), "Internet Engineering Task Force," [Online], https://self-issued.info/docs/draft-ietf-oauth-json-web-token.html, tanggal akses: 20-Jan-2018.

[15] "FIPS PUB 198-1. The Keyed-Hash Message Authentication Code (HMAC)," Federal Information Processing Standards Publication, hal. 13, 2008.

[16] T. Ramadhany, (2006), "Keyed-Hash Message Authentication Code (HMAC)," [Online], https://anzdoc.com/keyed-hash-message-authentication-codehmac.html, tanggal akses: 20-Jan-2018.

[17] K. I. Santoso, "Dua Faktor Pengamanan Login Web Menggunakan Otentikasi One Time Password Dengan Hash SHA," *Semin. Nas. Teknol. Inf. Komun. Terap. 2013*, 2013, hal. 204–210.

[18] B. Maryanto, "Penggunaan Fungsi Hash Satu-Arah Untuk Enkripsi Data," *Media Informatika*, Vol. 7, No. 3, hal. 1–10, 2008.