SEMINAR NASIONAL TEKNOKA

Vol. 6, 2021
ISSN No. **2502-8782**

# Implementasi Sistem Kriptografi RSA *Signature* dengan SHA-256 pada Mekanisme Autentikasi REST API

**Ilyas Mahfud[1)] & Putranto Hadi Utomo[2)]**
[1,2)]Universitas Sebelas Maret, Jl. Ir. Sutami No.36, Kentingan, Kec. Jebres, Kota Surakarta, 57126, Indonesia
E-mail: ilyasmahfud39@student.uns.ac.id; putranto@staff.uns.ac.id

***Abstrak***

*Komunikasi antar individu semakin mengalami peningkatan, maka dibutuhkan teknologi Application Programming Interface (API) untuk memfasilitasi pertukaran informasi. Teknologi API yang terkenal adalah Representational State Transfer (REST) dan keamanan pada REST memerlukan sistem autentikasi yang akan memberikan hak akses pada server REST. Proses autentikasi yang sering dilakukan adalah dengan verifikasi nama pengguna dan kata sandi sebagai muatan berbentuk Javascript Object Notation (JSON). Sebagai upaya pengamanan, kata sandi dapat dienkripsi dengan JSON Web Token (JWT) agar data tersamarkan ketika server diretas. Pada implementasi JWT, dibutuhkan sistem kriptografi pembuat tanda tangan digital untuk memastikan keotentikan pengguna. Pada Penelitian ini akan dibahas salah satu sistem kriptografi yang dapat diandalkan, yakni Rivest, Shamir, and Adleman (RSA) tanda tangan dengan Secure Hash Algorithm 256-bit (SHA-256) atau disingkat menjadi RS256. RS256 adalah algoritme asimetris yang menggunakan pasangan kunci publik dan kunci pribadi. Untuk mendapatkan kunci dari RSA perlu 3 proses yakni proses pembentukan, tanda tangan dan verifikasi kunci. Yang dimana saat proses tanda tangan dan verfikasi, digunakan fungsi hashing SHA-256 untuk menghindari resistensi tabrakan.*

***Kata kunci**: Informasi, REST, autentikasi, JWT, kriptografi, RS256*

***Abstract***

*Communication between individuals is increasing, so Application Programming Interface (API) technology is needed to facilitate the exchange of information. Well-known API technology is Representational State Transfer (REST), and the security in REST requires an authentication system to grant access to a REST server. The authentication process often used is by verifying the username and password as a payload as JavaScript Object Notation (JSON). For security, we can encrypt passwords with JSON Web Token (JWT) to disguise data when the server is hacked. It required a cryptographic system for creating digital signatures to ensure user authentication in this implementation. In this study, we will discuss one of the cryptographic systems, namely Rivest, Shamir, and Adleman (RSA) signature with Secure Hash Algorithm 256-bit (SHA-256) or abbreviated as RS256. RS256 is an asymmetric algorithm that uses public and private key pairs. There is a key generation, signature, and verification process to get a key from RSA. The SHA-256 hashing function is needed to avoid collision resistance in the signature and verification process.*

***Keyword**: Information, REST, authentication, JWT, cryptographic, RS256*

## 1 PENDAHULUAN

Arus pertukaran informasi kini semakin deras dan menjadi kebutuhan utama bagi banyak orang. Untuk memfasilitasi pertukaran informasi atau data antar individu dalam sebuah aplikasi, dibutuhkan teknologi *Application Programming Interface* (API). Menurut Kumari [1], salah satu teknologi API yang populer adalah *Representational State Transfer* (REST). REST adalah arsitektur standar atau gaya komunikasi yang menggunakan penamaan jalur API yang umum. REST adalah arsitektur komunikasi berbasis *client-server* standar yang banyak digunakan untuk mengembangkan *web service*. Namun keamanan pada REST API masih sangat lemah. Untuk mengatasi kekurangan ini, perlu sistem autentikasi untuk memberikan hak akses data pada *server* REST.

Pada tahun 2012, Gaol dan Evans [2] menjelaskan bahwa autentikasi memungkinkan pengguna yang berwenang untuk masuk ke jaringan dan pengguna yang tidak sah untuk masuk ke jaringan. Proses tersebut biasanya

*Dipresentasikan pada Tanggal 27 November 2021*

dilakukan dengan memverifikasi email dan kata sandi pengguna sebagai *payload* dalam bentuk *JavaScript Object Notation* (JSON) yang akan mengakses *server*. Kemudian JSON inilah yang menjadi format ringkas pertukaran data.

Dalam penelitian Bradly dan Sakimura [3], JSON *Web Token* (JWT) dijadikan solusi untuk menyamarkan kata sandi sebagai upaya tambahan keamanan. JWT adalah token dalam bentuk *string* panjang acak yang digunakan untuk proses autentikasi sistem dan pertukaran informasi. JWT mengamankan informasi dalam klaim yang dienkripsi berbentuk JSON dan menjadi dari JSON *Web Signature* (JWS). Selain JWT bisa memberikan keamanan dan kewenangan akses data dari *server* REST, JWT dipilih karena fitur *expiration*-nya yang dimana token akan berganti-ganti dan memerlukan *login* kembali untuk memastikan pengguna bukanlah robot.

Dalam implementasi JWT diperlukan suatu metode untuk menjaga kerahasiaan data/pesan, metode tersebut dinamakan dengan kriptografi. Lebih rincinya, kriptografi ini mempelajari teknik matematika yang berkaitan dengan aspek keamanan informasi, kerahasiaan, validitas, integritas, dan autentikasi. Salah satu kriptosistem paling terpercaya adalah RSA *signature* dengan SHA-256 (RS256). Dipilih RSA, karena RSA adalah algoritma asimetris yang menggunakan pasangan kunci publik-pribadi, dengan penyedia layanan punya kunci pribadi (rahasia) untuk membuat tanda tangan, dan konsumen JWT mendapatkan kunci publik untuk memvalidasi tanda tangan. Sementara SHA-256 adalah algoritma *hashing* searah yang dipilih agar data yang tersimpan memiliki kemungkinan kecil untuk didekripsi jika data bocor.

Pada penelitian ini akan dibangun sistem dengan arsitektur REST API yang berfokus pada sistem autentikasi dengan menggunakan JWT sebagai upaya untuk mengamankan data berbentuk klaim yang akan dienkripsi hingga berbentuk JSON dan menjadi muatan untuk melakukan proses autentikasi. Pada JWT pula akan diterapkan algoritma RS256 untuk menyamarkan kata sandi yang akan disimpan dalam *database*, sehingga jika ada kebocoran data, kata sandi akan tetap aman karena data yang bocor akan berbentuk *string* panjang acak yang tidak mudah didekripsi.

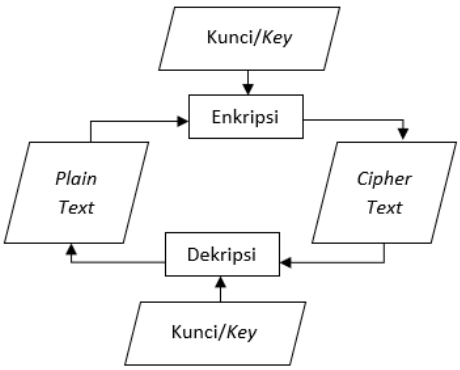
## 2 LANDASAN TEORI

Penelitian ini bertujuan menerapkan sistem kriptografi RS256 pada mekanisme autentikasi REST API. Oleh karena itu, tujuh definisi yang mendasari penelitian perlu diuraikan. Definisi tersebut meliputi pengertian dasar kriptografi, pengertian algoritma RS256, konsep RSA, konsep SHA-256, konsep REST API, konsep JWT, dan Java Spring Boot.

### 2.1 Kriptografi

Kriptografi menurut terminologi adalah ilmu dan seni dalam melindungi pesan. Munir [4] mendefinisikan kriptografi sebagai ilmu yang mempelajari teknik matematika yang berkaitan dengan aspek kemanan informasi seperti aspek kerahasiaan, integritas data, autentikasi pengirim atau penerima pesan, dan validitas pesan.

Data atau pesan dalam kriptografi disebut *plain text*. Kemudian proses menyamarkan data/pesan disebut enkripsi. Data yang dienkripsi disebut dengan *cipher text.* Kebalikan dari enkripsi atau proses pengembalian *cipher text* ke *plain text* disebut dekripsi. Dalam proses enkripsi dan dekripsi diperlukan kunci rahasia untuk mendapatkan *cipher text* dan begitu juga pada proses dekripsi.




***Gambar 1*** *Proses enkripsi dan dekripsi*

### 2.2 Algoritme RS256

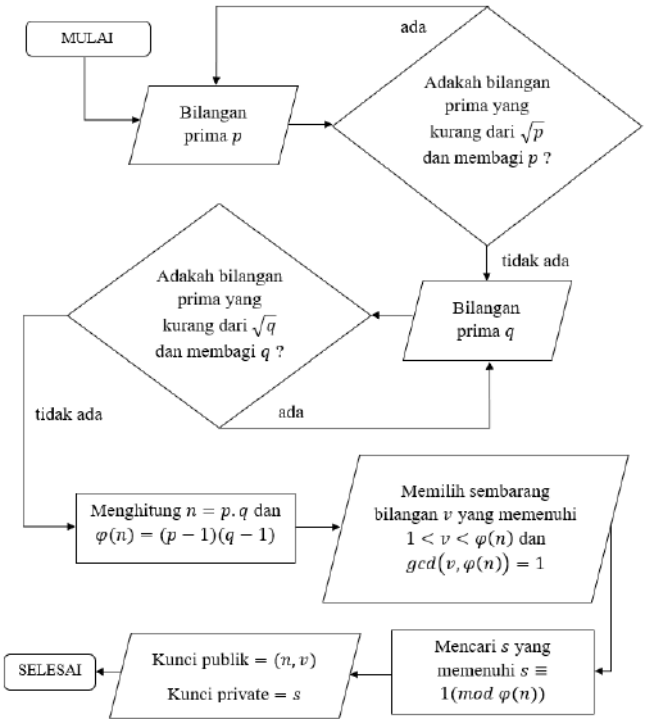
RS256 adalah algoritma asimetris yang menggunakan pasangan kunci publik-pribadi, dengan penyedia layanan memiliki kunci pribadi (rahasia) untuk menghasilkan *signature* atau tanda tangan, dan konsumen JWT mendapatkan kunci publik untuk memvalidasi tanda tangan.

### 2.3 Algoritma RSA

RSA adalah kepanjangan tiga nama orang yaitu: Rivest, Shamir, dan Adleman. Ketiga orang inilah yang mengenalkan tanda tangan digital RSA. Dalam penelitian Hoffstein *et al*. [5] dijelaskan bahwa tanda tangan digital RSA diperoleh melalui tiga proses yakni proses pembentukan kunci, proses tanda tangan digital, dan proses verifikasi. Algoritma pembuatan kunci RSA sebagai berikut.

1. Pilih bilangan prima pertama, misal $p$.
2. Cek apakah ada bilangan prima yang kurang dari $\sqrt{p}$ dan membagi habis $p$.
3. Jika ada bilangan prima, maka cari bilangan prima yang lain. Jika tidak ada, lanjutkan ke langkah selanjutnya.
4. Cari bilangan prima kedua, missal $q$.
5. Cek apakah ada bilangan prima lain yang kurang $\sqrt{q}$ dan membagi habis $q$.
6. Jika ada bilangan prima, maka cari bilangan prima yang lain. Jika tidak ada, lanjutkan ke langkah selanjutnya.
7. Kalikan kedua bilangan tersebut, $n = p.q$.
8. Hitung fungsi phi Euler dari $n$, $\varphi(n) = (p-1)(q-1)$.
9. Pilih bilangan bulat antara 1 dan $\varphi(n)$ yang saling prima dengan $\varphi(n)$ misal $v$.
10. Cek apakah bilangan tersebut saling prima dengan $\varphi(n)$ menggunakan algoritma Euclides. Jika $b = aq + r$ maka $gcd(b,a) = gcd(a,r)$.
11. Jika saling prima, maka selanjutnya adalah mencari $s = v^{-1}\, modulo\, \varphi(n)$. Jika tidak maka cari bilangan yang lain.
12. Mempublikasikan kunci publik $(n, v)$.
13. Merahasiakan kunci privat $s$ dan juga bilangan $p, q, \varphi(n)$.

*Flowchart* pembentukan kunci RSA disajikan dalam Gambar 2.

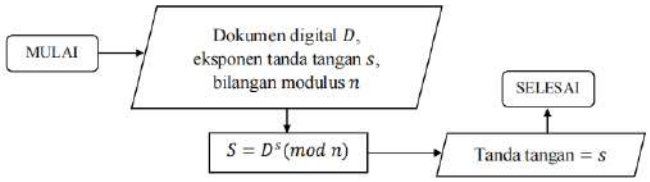



***Gambar 2*** *Flowchart pembentukan kunci RSA*

Algoritma pembuatan kunci RSA sebagai berikut.

1. Mencari dokumen digital dan mencari nilai *hash*-nya, misal $D$.
2. Mencari kunci publik $(n, v)$ dan kunci privat $s$.
3. Memangkatkan dokumen tersebut dengan kunci privat $s\ modulo\ n$. Jadi $S = D^s(mod\ n)$.
4. Bilangan $S$ adalah tanda tangan digital dari dokumen tersebut.

*Flowchart* penandatanganan RSA disajikan dalam Gambar 3.

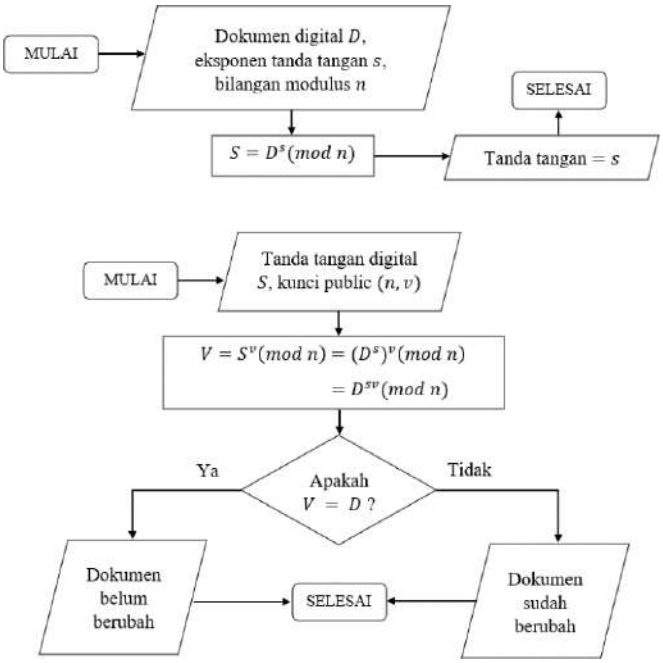



***Gambar 3*** *Flowchart penandatanganan RSA*

Algoritma verifikasi RSA sebagai berikut.

1. Cari tanda tangan digital $S$.
2. Cari kunci publik $(n, v)$.
3. Pangkatkan tanda tangan dengan $V = S^v(mod\ n) = (D^s)^v(mod\ n) = D^{sv}(mod\ n)$.
4. Jika $V = D$ berarti dokumen masih otentik, jika tidak berarti dokumen sudah berubah.

*Flowchart* verifikasi RSA disajikan dalam Gambar 4.




***Gambar 4*** *Flowchart penandatanganan RSA*

Proses mengubah kunci publik RSA menjadi *cipher text* bisa dilakukan dengan persamaan berikut.



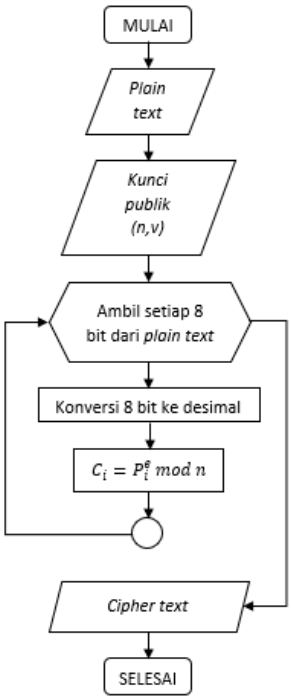
*ISSN No. 2502-8782*

$$C_i = P_i^e \ mod \ n \quad (1)$$

Dimana

$C$ : *Cipher text*
$P$ : *Plain text*
$e$ : Kunci publik
$n$ : Hasil perkalian $p$ dan $q$
$i$ : Urutan karakter

Flowchart mengubah kunci publik dalam bentuk angka yang diperoleh dari RSA menjadi *cipher text* disajikan dalam Gambar 5.




***Gambar 5*** *Flowchart mengubah plain text ke cipher text*

## 2.4 Algoritma SHA-256

*Secure Hash Algorithm* 256-bit (SHA-256) adalah fungsi hash yang sering dipakai agar tidak ada kemungkinan terjadi *collision resistance* (pemberian input data berbeda namun memiliki nilai hash yang sama). SHA-256 memiliki 8 langkah pengerjaan sebagai berikut.

1. Tambahkan bit *padding*

   Pada langkah pertama, pesan/data dalam biner, angka 1 dipersiapkan, lalu ditambahkan bit-bit pengganjal yakni angka 0 hingga panjang pesan kongruen dengan 448 modulo 512.
2. Panjang *append*

   Panjang pesan yang asli kemudian ditambahkan sebagai angka biner 64 bit. Setelah itu maka Panjang pesan menjadi kelipatan 512 bit.
3. *Parsing* data/pesan

   Data/pesan yang sudah di-*padding* lalu dibagi menjadi N buah blok 512 bit, $M^{(1)}$, $M^{(2)}$, ... , $M^{(n)}$, dengan menambahkan blok 64 bit.
4. Mempersiapkan jadwal

   Setiap blok 512 bit yang sebelumnya dipisah menjadi 16 kata 32 bit : $M_0^i, M_1^i, \dots, M_{15}^i$ yang kemudian akan diubah menjadi 64 kata 32 bit, kata-kata dari jadwal pesan diberi label $W_0, W_1, \dots, W_{63}$ dengan rumusan baku SHA-2 sebagai berikut.

$$W_t = \begin{cases} M_t^{(t)} & ,0 \le t \le 15 \\ \sigma_1^{256}(W_{i-2}) + W_{i-7} + \sigma_0^{256}(W_{i-15}) + W_{i-16} & ,16 \le t \le 63 \end{cases} \quad (2)$$

Dengan fungsi $\sigma_0$ dan $\sigma_1$dirumuskan sebagai berikut.

$$\sigma_1^{256}(W_{i-2}) = \left(ROTR^{17}(W_{i-2})\right) \oplus \left(ROTR^{19}(W_{i-2})\right) \oplus \left(SHR^{10}(W_{i-2})\right) \quad (3)$$

$$\sigma_0^{256}(W_{i-15}) = \left(ROTR^{7}(W_{i-15})\right) \oplus \left(ROTR^{18}(W_{i-15})\right) \oplus \left(ROTR^{3}(W_{i-15})\right) \quad (4)$$

Keterangan:

$W_t$ : Blok data/pesan baru
$M_t$ : Blok data/pesan lama
$W_{i-2}$ : Blok data/pesan dari $W$ ke $i-2$
$W_{i-15}$ : Blok data/pesan dari $W$ ke $i-15$
$ROTR$ : *Rotate right*
$SHR$ : *Shift right*
$\oplus$ : Operator $XOR$

$ROTR^n(x)$ adalah operasi geser kanan putar melingkar, dengan $x$ adalah sebuah penjadwalan pesan $(w)$ dan $n$ adalah bilangan bulan $(0 \le n < w)$, yang dapat didefinisikan $ROTR^n(x) = ((x \gg n) \vee (x \ll w-n)) = ROTR^{w-n}(x)$. Dalam hal ini $SHR^n(x)$ adalah operasi menggeser $x$ sebanyak $n$ posisi ke kanan.

5. Inisialisasi nilai *hash*

   Nilai *hash* awal, $H_0^{(0)} - H_7^{(0)}$ bersifat konstan. Nilai-nilai *hash* awal disajikan pada Tabel 1.

***Tabel 1*** *Nilai awal hash*

| Variabel | Hash value | Variabel | Hash value |
|---|---|---|---|
| $H_0^{(0)}$ | 6a09e667 | $H_4^{(0)}$ | 510e527f |

| | | | |
|---|---|---|---|
| $H_1^{(0)}$ | bb67ea85 | $H_5^{(0)}$ | 9b05688c |
| $H_2^{(0)}$ | 3c6ef372 | $H_6^{(0)}$ | 1f83d9ab |
| $H_3^{(0)}$ | a54ff53a | $H_7^{(0)}$ | 5be0cd19 |

6. Inisialisasi 8 variable kerja $a, b, c, d, e, f, g,$ dan $h$ $(i-1)$ untuk $t = 0$ hingga $t = 63$

Algoritma tersebut akan bekerja dengan perulangan 64 kali perhitungan untuk setiap blok. Dengan nilai awal untuk kedelapan variabel tersebut sebagai berikut.

$$
\begin{aligned}
T_1 &= h + \sum\nolimits_1^{(256)}(e) + Ch(e,f,g) + K_1^{(256)} + W_t \\
T_2 &= \sum\nolimits_0^{(256)}(a) = Maj(a,b,c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

Dengan fungsi $\sum_1^{(256)}(e)$, $\sum_0^{(256)}(a)$, $Ch$, $Maj$ dirumuskan sebagai berikut.

$$
\begin{aligned}
Ch\,(e,f,g) &= (x\text{\^{}}y) \oplus (\sim e\text{\^{}}g) \\
Maj(a,b,c) &= (a\text{\^{}}b) \oplus (a\text{\^{}}c) \oplus (b\text{\^{}}c) \\
\sum\nolimits_0^{(256)}(e) &= ROTR^2(e) \oplus ROTR^{13}(e) \oplus ROTR^{22}(e) \\
\sum\nolimits_1^{(256)}(a). &= ROTR^6(a) \oplus ROTR^{11}(a) \oplus ROTR^{25}(a)
\end{aligned}
$$

Keterangan:

| | |
|---|---|
| $a, b, c, d, e, f, g, h$ | : Peubah tipe heksadesimal |
| $K_1^{(256)}$ | : Konstanta SHA-256 |
| $ROTR$ | : *Rotate right* |
| $\oplus$ | : Operator $XOR$ |
| ^ | : Operator $AND$ |

***Tabel 2*** *Konstanta SHA-256*

| | | | |
|---|---|---|---|
| 428a2f98 | 71377791 | b5c0fbcf | e9b5dba5 |
| 3956c25b | 59f111f1 | 923f82a4 | ab1c5ed5 |
| d807aa98 | 12835b01 | 243185be | 550c7dc3 |
| 72be5d74 | 80deb1fe | 9bdc06a7 | c19bf174 |
| e49b69c1 | efbe4786 | 0fc19dc6 | 240ca1cc |
| 2de92c6f | 4a7484aa | 5cb0a9dc | 76f988da |
| 983e5152 | a831c66d | b00327c8 | bf597fc7 |
| c6e00bf3 | d5a79147 | 06ca6351 | 14292967 |
| 27b70a85 | 2e1b2138 | 4d2c6dfc | 53380d13 |
| 650a7354 | 766a0abb | 81c2c92e | 92722c85 |
| a2bfe8a1 | a81a664b | c24b8b70 | c76c51a3 |
| d192e819 | d6990624 | f40e3585 | 106aa070 |
| 19a4c116 | 1e376c08 | 2748774c | 34b0bcb5 |
| 3 91c0cb3 | 4ed8aa4a | 5b9cca4f | 682e6ff3 |
| 74 8f82ee | 78a5636f | 84c87814 | 8cc70208 |
| 90 befffa | a4506ceb | bef9a3f7 | c67178f2 |

7. Menghitung hasil akhir dengan menjumlah semua variabel dan nilai awal *hash* $H^i$

$$
\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)} \\
H_1^{(i)} &= b + H_1^{(i-1)} \\
H_2^{(i)} &= c + H_2^{(i-1)} \\
H_3^{(i)} &= d + H_3^{(i-1)} \\
H_4^{(i)} &= e + H_4^{(i-1)} \\
H_5^{(i)} &= f + H_5^{(i-1)} \\
H_6^{(i)} &= g + H_6^{(i-1)} \\
H_7^{(i)} &= h + H_7^{(i-1)}
\end{aligned}
$$

8. Output

Setelah melakukan perulangan langkah 1 hingga 4 sebanyak N kali, fungsi *hash* yang dihasilkan adalah sebagai berikut.

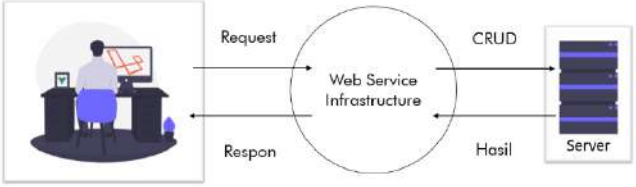
$$H_0^N \left|\left| H_1^N \right|\right| H_2^N \left|\left| H_3^N \right|\right| H_4^N || H_5^N || H_6^N || H_7^N$$

## 2.5 REST API

REST atau *Representational State Transfer* adalah standar arsitektur atau gaya komunikasi yang menggunakan penamaan jalur API yang menggunakan protokol *Hypertext Transfer Protocol* (HTTP) untuk mengirim dan menerima data. Metode dalam HTTP yang sering dipakai pada REST API sebagai berikut.

1. Metode *Get* dipakai untuk mendapatkan data dari *server* REST.
2. Metode *Post* dipakai untuk menciptakan data di *server* REST.
3. Metode *Put* dipakai untuk memperbarui data di *server* REST.
4. Metode *Delete* dipakai untuk menghapus data dari *server* REST.

Ada 2 tipe arsitektur REST, yaitu REST *server* sebagai penyediakan data dan REST *client* sebagai yang mengakses dan mengambil data. Tiap data akan diidentifikasi menggunakan *Uniform Resource Identifier* (URI) *link* dan REST *server* merespon dengan mengirimkan sebuah HTTP *response* yang sesuai dengan permintan REST *client*.



***Gambar 6** Arsitektur REST*

Mekanisme sederhana dari sebuah proses REST diawali dengan adanya sebuah REST *server* sebagai penyedia data. Kemudian REST *client* sebagai pembuat HTTP *request* ke *server* melalui sebuah URI *link*. Adapun bagian-bagian dari HTTP *request* sebagai berikut.

1. Salah satu metode HTTP yang sudah dijelaskan sesuai kebutuhan.
2. URI *link* sebgai penunjuk lokasi data di *server*.
3. *Request header* yang berisi *authorization*, tipe *client* dan yang lainnya.
4. *Request body* yang berisi data yang diberikan *client* ke *server* seperi parameter URI *link*.

Setelah *server* menerima HTTP *request*, akan dibalas dengan HTTP *response* sesuai permintaan *client*.

Adapun bagian-bagian HTTP *response* sebagai berikut.

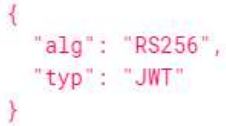
1. *Response code* yang berisi status *server* pada permintaan *client* seperti 200, 401, 404 dan lainnya,
2. *Response header* yang berisi tipe konteks, *cache tag* dan yang lainnya.
3. *Repsonse Body* yang berisi data yang disediakan server.

Kemudian *response* dari *server* akan ditampilkan kepada pengguna dalam bentuk tampilan web ataupun *mobile*.

## 2.6 JSON *Web Token* (JWT)

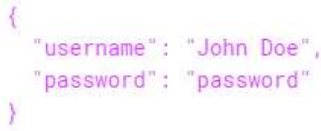
Menurut Bradley dan Sakimura [3], JWT adalah format standar dalam mengamankan informasi pribadi menjadi sebuah klaim yang dienkripsi hingga berbentuk JSON dan menjadi muatan untuk JSON *Web Token* (JWT). Klaim akan terlindungi dengan adanya tanda tangan digital seperti *Message authentication code* (MAC) atau dienkrip. Karakteristik dari JWT sebagai berikut.

1. *Header*
   Ada dua bagian dari *header*, yaitu: jenis token dan algoritma *hashing* yang digunakan, seperti HMAC SHA256 atau RSA.



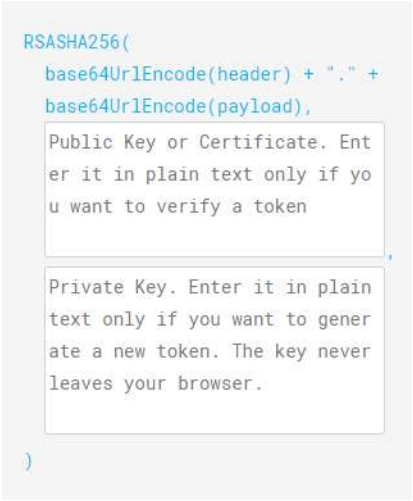

***Gambar 7** Header pada JWT*

2. *Payload*
   *Payload* berisi klaim yang menyatakan tentang suatu entitas seperti nama pengguna dan kata sandi.



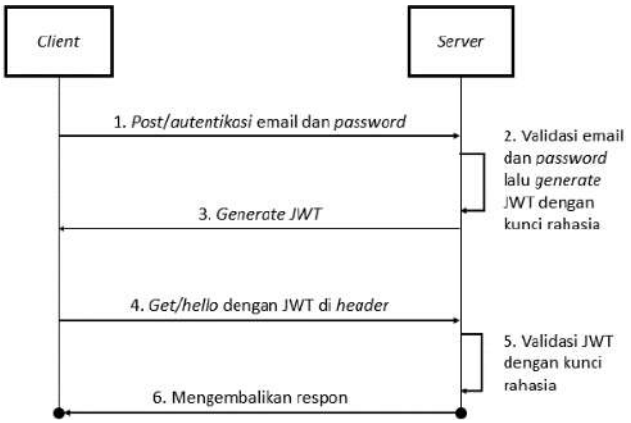

***Gambar 8** Payload pada JWT*

3. *Signature*
   Dalam membuat tanda tangan, perlu mengambil data pada *header* yang berupa *secret key* dan algoritma yang ditentukan dalam *header*.




***Gambar 9** Signature pada JWT*

JWT adalah format standar dalam mengamankan data menjadi sebuah klaim yang akan dienkripsi ke dalam bentuk JSON dan menjadi muatan dari JSON *Web Signature* (JWS). Klaim atau token dari JWT berbentuk *string* yang panjang dan acak untuk melakukan autentikasi dan pertukaran data. Token JWT dapat dikirim melalui URI *link*, parameter HTTP *post* atau di dalam *header* HTTP. Token tersebut juga memiliki ukuran yang kecil, sehingga dapat ditransmisikan dengan cepat. Mekanisme autentikasi dengan JWT disajikan pada Gambar 10.




***Gambar 10*** *Proses autentikasi pada JWT*

### 2.7 Java Spring Boot

Salah satu *framework* Java *open source* yang menyediakan infrastruktur lengkap adalah Spring. Tujuan Spring adalah membangun logika bisnis dengan cepat dan mudah. Kerangka kerja yang dikembangkan oleh Rod Johnson pada tahun 2003 ini, membantu mengembangkan sistem modular karena dapat digunakan kembali dan menerapkan pemrograman berorientasi objek. Kemudian muncul *framework* Spring Boot, yang merupakan evolusi dari kerangka Spring. Spring Boot juga membantu mempercepat pengembangan layanan web karena anotasinya dapat mempercepat otomatisasi yang dikonfigurasi oleh Spring Boot. Sehingga pengembangan suatu sistem dapat lebih fokus pada pengembangan fitur bisnis dan kurang pada infrastruktur [11].

## 3 METODOLOGI PENELITIAN

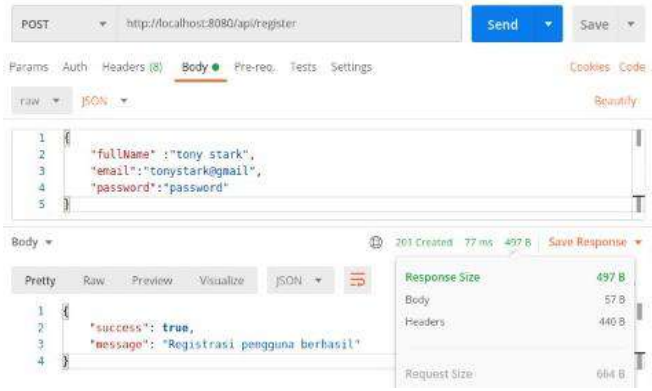
Dalam melakukan penelitian digunakan metode kajian pustaka dan *system development life cycle* (SDLC). Tinjauan pustaka dilakukan terhadap sumber berupa buku, jurnal, maupun skripsi mengenai sistem kriptografi RS256 dan mekanisme autentikasi REST API dengan menggunakan JWT. SDLC adalah pendekatan sistematis dalam implementasi sistem yang dimulai dengan analisis, desain, pengkodean dan pengujian. Langkah-langkah operasional yang dilakukan dalam penelitian ini adalah sebagai berikut.

1. Menganalisis penelitian terkait algoritma RS256.
2. Menganalisis algoritma kriptografi RS256 dalam JWT serta teknik-teknik yang digunakan.
3. Mendesain arsitektur *server* REST API. Arsitekturnya mencakup lapisan *middleware*, lapisan logika bisnis dan lapisan basis data.
   a. Lapisan *middleware* bertugas mengartikan setiap permintaan dan memvalidasinya.
   b. Lapisan logika bisnis bertugas sebagai tempat untuk implementasi fungsionalitas inti sistem.
   c. Lapisan basis data bertugas mengelola data berdasarkan perintah sistem. Data dan informasi nantinya akan di berikan dalam bentuk JSON.
4. Melakukan pengujian dan analisis pada sistem yang dibuat untuk diambil kesimpulan dari hasil penerapannya.

## 4 HASIL DAN PEMBAHASAN

Penelitian ini menghasilkan sebuah *web service* dengan proses autentikasi pada REST API nya menggunakan algoritma RS256, sehingga password dapat terenkripsi dengan aman di*server*.

Pengujian pada penelitian ini menggunakan postman sebagai REST *client* yang digunakan untuk menguji REST API yang sudah dibuat. Parameter dalam pengujian ini adalah kecepatan dalam enkripsi kata sandi. Pengujian ini dilakukan dengan 2 proses yaiut POST *register* dan POST *login*. POST *register* untuk melakukan enkripsi kata sandi yang dilakukan dengan melakukan registrasi pengguna. Kemudian proses POST *login* digunakan untuk mengirimkan parameter nama pengguna dan kata sandi untuk menghasilkan token autentikasi. Saat melakukan *request* berupa *register*, kata sandi yang berupa *plain text* akan dienkripsi menjadi *cipher text* dan disimpan ke *server*. Pada penelitian ini digunakan *payload register* berbentuk JSON yang berisi *fullname*, *email* dan *password*. Proses enkripsi dari POST *register* dengan postman dapat dilihat pada Gambar 11.



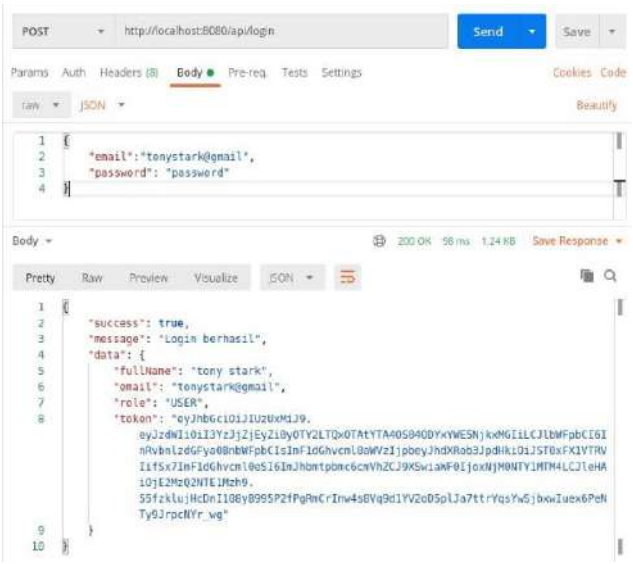

***Gambar 11*** *POST register dengan postman*

Pada tahap uji coba, kami melakukan proses enkripsi kata sandi kepada sebanyak 11 pengguna, dengan *payload* kata sandi atau *plain text*-nya sama untuk semua pengguna yakni “password”. Hasil enkripsi kata sandi pengguna disajikan pada Gambar 12.

| email<br>character varying (255) | password<br>character varying (255) |
|---|---|
| tonystark@gmail.com | $2a$10$C0GnhsXBSY/YhK5oU4A4auZi4YCilt7QulRnoLE/PouA8AKa... |
| brucewayne@gmail.com | $2a$10$sbSJ1jVLZqNtIbm8.LZ2.0vCbm9IrXFYk.wu4AP3Yv6r01JP3... |
| flash@gmail | $2a$10$JCKnOR.BROBufzrI2FSJYu1YfJTu/3Fk7OnvIE/CXy1uCFXUf7... |
| gundala@gmail | $2a$10$WHvTXdyT/HTSS8MDS.TICe/VMx4RWJXtlmd.RQNUqLE2c1... |
| wiro_sableng@gmail | $2a$10$g8mVLh0/fTA9IJtQoMUfvO5LkEhOiz10tj/58ct2dtrPS/OFNY... |
| spiderman@gmail | $2a$10$04Xmjz94EmUTtyMPGm/tjOUoL.SFmV4BqJtED1WKaXn6M... |
| herkules@gmail | $2a$10$0CUKEANVlbhFETSCNly5T060DbuFlvANfkGLgKyOHEKx0m... |
| wonder_woman@gmail | $2a$10$NCrq8YbUpk6uvUHeRZO1OO76QgCLS1VL4OUrzPJOgpl7.KF... |
| archiles@gmail | $2a$10$/bbyJjIEg2sM66sycEH5Q.PRxgOlEUIX68/Y/ ICdWXgFKZ0elx... |
| gojo@gmail | $2a$10$apxANhP3ByM1mi6mUOzUnumgGfdLB5RbOIfwbTdgVTjdSE... |

***Gambar 12*** *Hasil enkripsi password*

Ketika pengguna melakukan POST *login*, secara otomatis token akan dikirimkan ke *header* HTTP. Misal kita gunakan *email* "tonystark@gmail" dan kata sandinya "password", maka akan didapatkan token dan kunci publik nya. Proses POST *login* ini membutuhkan waktu 72 ms, sebagaimana yang disajikan pada Gambar 13.




***Gambar 13*** *POST login dengan postman*

Bentuk token bisa dilihat seperti pada gambar diatas dan kunci publik yang didapatkan dari proses pembangkitan JWT dengan algoritma RS256 disajikan pada Gambar 14.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtcUbJDyqtRAcGWBgLirg\ngnjlp
KiLUqombtri0ciqd1go85Q8ruKlgJVcfJqWvjdyiPIPfh87CiWVVyDP8elw\n57uhuOXuHM
hB87iukwulqNxfDq+jvO73BVgUJ8tK8ZwPgVgOz47AdOV210kwtW2X\nnfihR0PotjPqzjK
svFG52ADcmIHBMXQ90vR24FmCj2a0C0e9Yb8ICuuefM+MYjtf\nBXFNY0xM7so2f7m3R0xj
3VzEVK5DNAHHAB7txoJrq0/oFSHS/kb+cM0tInFZBrAS\nNUe86uiuBvXXD3cJpsr+Bu+Aa
+3d8T0t8Ce3WHza2gYvGhqLt59xZRHpV0lgab0G\nbQIDAQAB
-----END PUBLIC KEY-----
```

***Gambar 14*** *Kunci publik RSA*

Lebih lanjut, token dan kunci publik yang pengguna peroleh dapat digunakan untuk mengakses API lain dari *web service* yang telah dibuat. Untuk melihat kinerja enkripsi pada REST API dengan menggunakan RS256, kami telah melakukan uji coba dengan melakukan proses POST *register* dengan 10 data pengguna. Hasil dari uji coba penelitian ini disajikan pada Tabel 3.

***Tabel 3*** *Hasil uji coba enkripsi RS256*

| Urutan | Pengguna | Waktu (ms) |
|---|---|---|
| 1 | Gojo | 297 |
| 2 | Archiles | 82 |
| 3 | Wonder Woman | 82 |
| 4 | Herkules | 79 |
| 5 | Spiderman | 77 |
| 6 | Gundala | 81 |
| 7 | Flash | 77 |
| 8 | Bruce Wayne | 76 |
| 9 | Tony Stark | 77 |
| 10 | Wiro Sableng | 77 |

Dengan kata sandi yang sama yakni "password", terlihat dalam Tabel 3 tersebut bahwa proses enkripsi membutuhkan waktu yang sedikit lebih lama ketika pertama kali REST API digunakan. Setelah itu proses enkripsi mulai menunjukkan waktu yang stabil di angka antara 76-82 ms.

## 5 SIMPULAN

Berdasarkan pembahasan dan uji coba *web service* yang terlah dilakukan, diperoleh keimpulan sebagai berikut.

1. Dengan implementasi sistem kriptografi RS256 pada mekanisme autentikasi REST API, kata sandi pengguna bisa terenkripsi menjadi *string* panjang acak. Sehingga jika ada kebocoran data, kata sandi akan tetap aman.
2. Dengan fungsi *hashing* SHA-256 yang digunakan pada RSA menjadikan kata sandi yang pengguna masukan walaupun sama tapi tetap akan terenkripsi menjadi *cipher text* yang berbeda dan memiliki panjang yang sama.
3. Dengan uji coba 10 pengguna, proses enkripsi kata sandi membutuhkan waktu antara 76-82 ms. Namun pada awal proses enkripsi, *web service* membutuhkan waktu hamper 4 kali lebih dari waktu-waktu setelahnya. Hal ini terjadi karena *web service* belum tersimpan pada memori di lokal komputer, sehingga memerlukan waktu yang lebih lama untuk mengeksekusi program diawal uji coba tersebut.

## KEPUSTAKAAN

[1] Kumari, V., *Web Services Protocol : SOAP vs REST*, IJARCET, Vol. 4, No. 5, pp. 2467–2469. (2015).

[2] Gaol , L. & Evans B., *Implementasi dan Analisis Authentikasi Jaringan Wireless Menggunakan Metode Extenxible Authentication Protocol - Transport Layer Security (EAP-TLS)*, JELIKU (Jurnal Elektronik Ilmu Komputer Udayana), Vol. 1 No. 1, 2012.

[3] M. Jones, J. Bradley, & N. Sakimura, *JSON Web Token (JWT)*, The RFC series. (2015).

[4] Tanaem, P.F., D. Manongga, & A. Iriani, *RESTFul Web Service Untuk Sistem Pencatatan Transaksi Studi Kasus PT. XYZ*, Jurnal Teknik Informatika dan Sistem Informasi, vol. 2, no. 1. (2016).

[5] Dermawan, D.C., *Implementasi Algoritme RSA (Rivest-Shamir-Adleman) pada Layanan Chatting Berbasis LAN (Local Area Network)*, Undergraduate thesis, Universitas Muhammadiyah Jember. (2015).

[6] Soram, R. & E. S. Meitei, *On the Performance of RSA in Virtual Banking*, International Symposium on Advanced Computing and Communication (ISACC), pp. 352-359. (2015).

[7] N. Agani, M. Hardjianto, & D. Virgian, *Pengamanan Sistem Menggunakan One Time Password Dengan Pembangkit Password Hash SHA-256 dan Pseudo Random Number Generator (PRNG) Linear Congruential Generator (LCG) di Perangkat Berbasis Android*, Budi Luhur Information Technology, vol. 13, no. 1. (2016).

[8] Munir, R., *Diktat Kuliah Kriptografi*, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung. (2006).

[9] Hoffstein, J., J. Phiper, & J.H. Silverman, *An Introduction to Mathematical Cryptography*, USA: Springer. (2014).

[10] Sklavos, N. & Koufopavlou, O., *Implementation of the SHA-2 Hash Family Standard Using FPGAs*, Journal of Supercomputing vol.31,pp. 227–248 (2005).

[11] Spring Boot, 2021, *Building an Application with Spring Boot*, https://spring.io/guides/gs/spring-boot/, diakses pada 1 September 2021. (2021).