Penggunaan Kriptografi pada JWT (JSON Web Token) dalam Implementasi Keamanan API

Rio Chandra Rajagukguk
Teknik Informatika, STEI
Institut Teknologi Bandung
Bandung, Indonesia
13514082@std.stei.itb.ac.id

***Abstract*—Pada *paper* ini dijelaskan penerapan kriptografi pada JWT (JSON *Web Token*) dalam implementasi keamanan komunikasi API antara *server* dan *client*. JWT merupakan salah satu standar industri saat ini dalam melakukan komunikasi API. Permasalahan komunikasi API adalah server tidak dapat mengenali pengguna yang melakukan *request* merupakan pengguna yang memiliki ijin atau tidak, permasalahan ini disebut dengan CORS *issue*. CORS dapat diselesaikan menggunakan berbagai cara salah satunya dengan autentikasi JWT. JWT akan memberikan sebuah kunci *hash* yang dijadikan sebagai pengenal kepada server setiap kali mengakses server tersebut. Hasil implementasi makalah ini menunjukkan seperti apa bentuk JWT diimplementasi pada sebuah server dalam skala eksperimen.**

Keywords—jwt; hash; sha256; kriptografi; api; rio

I. INTRODUCTION (*HEADING 1*)

API (*Application Programming Interface*) merupakan salah satu bentuk komunikasi yang banyak digunakan saat ini pada sistem *client-server* baik pada aplikasi *mobile*, web, maupun *dekstop*. Berbeda dengan menampilkan web, API tidak menampilkan halaman web tetapi hanya menampilkan informasi yang dibutuhkan saja berupa teks pada umumnya. Komunikasi API biasanya dilakukan antara *server* dan *client* melalui salah satu protokol TCP/IP yaitu HTTP/HTTPS. Salah satu penerapan API yang populer dengan mengirimkan data berupa teks dalam format JSON (*JavaScript Object Notation*) oleh *server* ke *client* atau sebaliknya. JSON adalah suatu format ringkas dalam melakukan pertukaran data komputer. Formatnya berbasis teks yang dapat dibaca dan dimengerti manusia serta digunakan untuk merepresentasikan struktur data sederhana dan larik asosiatif yang disebut objek. Format JSON sering digunakan untuk mentransmisikan data terstruktur melalui suatu koneksi jaringan pada suatu proses yang disebut serialisasi (*serialization*). Aplikasi utamanya adalah pada pemrograman aplikasi berbasis API dengan berperan sebagai alternatif terhadap penggunaan tradisional format XML. Proses pengiriman data melalui API dapat dilakukan oleh setiap orang yang melalui permintaan (*request*) API ke sebuah server, pada kebanyakan kasus server harus dapat mengidentifikasi apakah permintaan tersebut dari pihak yang telah diberi hak untuk mendapatkan data yang diinginkan atau tidak. Sehingga, salah satu permasalahan dalam sistem API menggunakan JSON berada pada pengamanan data yang dikirimkan untuk mengidentifikasi pihak yang melakukan permintaan (*request*). Permasalahan ini memiliki banyak solusi salah satunya JWT (JSON *Web Token*), secara sederhana JWT terdiri dari dua proses utama yaitu proses meminta autentikasi dan proses permintaan data. Awalnya proses permintaan kunci, pihak yang ingin mengakses *resources* pada server, terlebih dahulu harus mengidentifikasikan diri atau yang disebut dengan proses autentikasi dengan mengikuti proses autentikasi yang disediakan oleh server tersebut, misalnya menggunakan *username* dan *password*. Pada saat *server* mengidentifikasi *credential* autentikasi pihak peminta akses merupakan pihak yang memiliki ijin, server akan mengembalikan sebuah kunci hash untuk digunakan berikutnya oleh pihak tersebut untuk mengakses server tersebut. Proses pembentukan *hash* tersebut dilakukan menggunakan algoritma *hash* standar yang ada di internet saat ini seperti yang terkenal yaitu MD5 dan SHA. Pada makalah ini akan diberikan batasan contoh penggunaan JWT *Token* menggunakan algoritma *hash* SHA yaitu lebih spesifiknya SHA-256 yaitu SHA dengan keluaran 256 bit.

II. LANDASAN TEORI

A. Kriptografi

Kriptografi adalah suatu ilmu dan seni untuk menjaga keamanan pesan (Schneier, 1996). Dalam kriptografi terdapat dua konsep utama yaitu enkripsi dan dekripsi. Enkripsi adalah proses mengubah teks awal (*plaintext*) menjadi teks tersandikan (*chipertext*), sedangkan dekripsi adalah proses sebaliknya mengubah *chipertext* menjadi *plaintext*. *Plaintext* adalah pesan awal teks yang ingin di enkripsi sedangkan *chipertext* adalah teks yang sudah tidak memiliki makna yang merupakan representasi dari *plaintext*.

Algoritma kriptografi berdasarkan kunci yang digunakan, dibedakan menjadi dua yaitu algoritma simetris dan algoritma asimetris. Algoritma simetris adalah algoritma kriptografi dimana untuk proses enkripsi dan dekripsi menggunakan kunci yang sama yaitu kunci privat. Dengan menggunakan algoritma ini, pengirim pesan dan penerima pesan terlebih dahulu harus menyepakati kunci privat yang akan digunakan sebelum nantinya mengirim pesan. Contoh algoritma simetris yang terkenal adalah DES, Rinjdael, Blowfish, IDEA, dan GOST. Algoritma asimeteris atau algoritma kunci publik adalah algoritma kriptografi dimana untuk proses enkripsi dan dekripsi menggunakan kunci yang berbeda yaitu kunci publik untuk

enkripsi dan kunci privat untuk dekripsi. Contoh algoritma asimetris yang terkenal adalah algoritma RSA, Rabin, ElGamal, DSA, dan ECC.

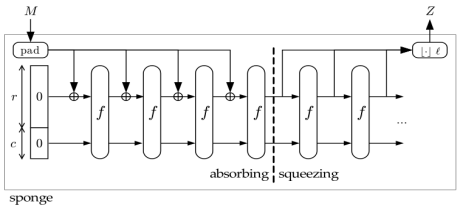
Selain algoritma kunci, terdapat jenis algoritma kriptografi lainnya yaitu fungsi *hash*. Fungsi hash yaitu fungsi yang menerima masukan sebuah teks lalu dilakukan operasi hash dengan keluaran berupa string yang panjangnya tetap untuk berapapun panjang teks masukan yang diberikan. Hasil keluaran fungsi hash disebut *message digest*. *Message digest* yang dihasilkan oleh fungsi hash tidak dapat dikembalikan lagi menjadi pesan semula (*irreversible*). Fungsi hash bersifat satu arah karena tidak bisa mengembalikan message digest ke teks awal. Fungsi *hash* dalam kriptografi dapat digunakan untuk menjaga keaslian data dengan cara menghitung nilai hash sebuah data. Jika data berubah, maka nilai hashnya juga akan berubah. Perubahan sedikit saja dalam data dapat mengakibatkan nilai *hash* yang berubah drastis. Dalam kriptografi, dikenal beberapa fungsi *hash* antara lain MD2, MD4, MD5, SHA-0, SHA-1, RIPEMD, WHIRLPOOL, dan lain – lain. Salah satu sifat yang ingin dicapai dalam sebuah fungsi *hash* H adalah untuk setiap *string* masukan x, tidak mungkin ada *string* y, $x \neq y$ sedemikian sehingga $H(x) = H(y)$. Namun, dalam kenyataannya pada beberapa algoritma fungsi *hash* terdapat kolisi, yaitu keadaan dimana dua buah *string* sembarang memiliki nilai *hash* yang sama.

B. Fungsi Hash NIST (SHA-3)

Seperti sejarah AES sebelumnya, *National Institute of Standards and Technology* (NIST) menyelenggarakan kompetisi terbuka untuk mengembangkan fungsi *hash* yang baru, bernama SHA-3. SHA-3 menjadi komplementer darj SHA-1 dan SHA-2. Kompetisi ini mulai diumumkan pada tahun 2007 dan berakhir pada Oktober 2012 dengan memilih pemenang sesuai kriteria yang telah ditetapkan.

Proses pemilihan terdiri dari dua putaran awal dan babak final. Jumlah *submission* adalah 64 rancangan fungsi *hash*. Pada putaran pertama (penyisihan) dipilih 51 *submission* terbaik, lalu pada putaran kedua (semi final): dipilih 14 *submission* terbaik dan berikutnya dilakukan babak final sebanyak 5 *finalis*.

Finalis terpilih menjadi pemenang dan menjadi algoritma untuk SHA-3 adalah keccak. Nama 'Keccak' berasal dari 'Kecak', tarian Bali. Keccak berbeda dari finalis SHA3 lainnya dalam hal menggunakan konstruksi 'spons' (*sponge construction*). Jika desain lainnya bergantung pada fungsi kompresi, keccak menggunakan fungsi non-kompresi untuk menyerap dan kemudian 'memeras' *digest* singkat. Jika dilihat sekilas tidak terlalu jelas, apakah desain ini lebih baik atau lebih buruk daripada pendekatan yang ada, tapi keccak berbeda dengan yang lain. NIST merasa bahwa dalam kasus kompetisi ini, yang berbeda adalah yang lebih baik.



sponge

Gambar 1. Konstruksi Spons pada Keccak

Proses konstruksi Spons pada algoritma keccak dapat dilihat pada Gambar 1. Secara umum, dalam konstruksi spon terdapat dua fase, yaitu fase *absorbing* dan fase *squeezing*.

1. Fase absorbing, merupakan fase dimana proses dilakukan terhadap semua *pecahan* dari *input* masukan dan di XOR-kan dengan bagian *bitrate* dari *state* kemudian dilewatkan kedalam fungsi f.

2. Fase *squeezing*, merupakan fase untuk mendapatkan hasil keluaran. Dalam fase ini dilakukan konkatenasi terhadap sejumlah bit tertentu dari hasil fungsi f sehingga jumlah bit konkatenasi tersebut sama dengan jumlah bit konkatenasi yang diinginkan.

C. JSON

JSON (*JavaScript Object Notation*) adalah suatu format ringkas pertukaran data komputer. Formatnya berbasis teks dan terbaca-manusia serta digunakan untuk merepresentasikan struktur data sederhana dan larik asosiatif (disebut objek). Format JSON sering digunakan untuk mentransmisikan data terstruktur melalui suatu koneksi jaringan pada suatu proses yang disebut serialisasi. Aplikasi utamanya adalah pada pemrograman aplikasi web AJAX dengan berperan sebagai alternatif terhadap penggunaan tradisional format XML.

Pada tujuan awal JSON merupakan *subset* bahasa pemrograman JavaScript (secara spesifik, edisi ketiga standar ECMA-262, Desember 1999) dan umumnya digunakan dengan bahasa tersebut, JSON dianggap sebagai format data yang tak tergantung pada suatu bahasa. Kode untuk pengolahan dan pembuatan data JSON telah tersedia untuk banyak jenis bahasa pemrograman. Format JSON dispesifikasikan di RFC 4627 oleh Douglas Crockford. Tipe media Internet resmi JSON adalah *application/json* sedangkan ekstensi berkasnya adalah .json.

D. CORS (Cross-Origin Resource Sharing)

CORS merupakan sebuah istilah untuk berbagi *resources* yang ada pada sebuah server kepada *request* yang berasal dari luar server tersebut. CORS sudah ada sejak awal internet dan merupakan hal yang lumrah saat awal-awal adanya internet. Namun seiring berkembangnya internet CORS menjadi satu issue tersendiri dalam berbagi *resources* kepada permintaan (*request*) yang tidak dikenal.

Server saat ini biasanya melakukan pembatasan terhadap *request* yang diterima kepada pihak-pihak tertentu saja, misal hanya kepada orang-orang yang terdaftar pada jaringan tersebut,

```
h_b64       = abcABC
p_b64       = defDEF
signature   = 1X2Y3Z
```

atau pada metode *request* tertentu saja. Proses permintaan ini membutuhkan sebuah pengenal khusus dari pihak yang melakukan permintaan. Ketika permintaan ditolak dengan alasan pihak yang melakukan permintaan tidak diberi ijin, maka akan mengembalikan *error* berupa CORS *denied*. CORS *denied* menyatakan *resources* yang diminta oleh pihak tersebut tidak diijinkan.

E. JSON Web Token

Issue CORS membutuhkan solusi berupa pengenalan pada *request* yang diterima oleh server. Untuk melakukan pengenalan terdapat satu solusi sederhana yaitu dengan memberikan *token* pengenal bagi pihak yang melakukan *request*. Token tersebut kemudian diberikan sebagai kunci setiap kali melakukan *request* ke server tersebut. *Token* tersebut tentutnya harus seoptimal mungkin menghemat proses kerja server dan tetap aman.

JWT merupakan sebuah implementasi solusi *token* web untuk masalah *resources sharing* pada domain berbeda (CORS). Pihak yang melakukan *request* tidak perlu menyimpan *session* untuk dapat tetap dikenali oleh server, dengan JWT, *request* yang dikirimkan cukup disisipkan pengenal JWT tersebut. JWT merupakan sebuah *token* yang dibuat dalam format JSON untuk tujuan Web (http/s) *request*.

Implementasi JWT paling banyak dilakukan pada server API untuk *multiplatform* seperti android, web, ios, desktop, dan lainnya. JWT terdiri dari lima proses utama yaitu:

1. Membuat *Header*, memberikan spesifikasi algoritma yang digunakan pada JWT tersebut berupa sebutan "*header*".

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

2. Membuat *Payload*, selain *header*, komponen lain yang dibutuhkan adalah identifikasi pengguna, misalkan *username* dan *password*

```
{
        "username": "kripto",
        "password": "12345"
}
```

3. Pembentukan Tanda Tangan (*Signature*), kedua komponen diatas lalu dibentuk sebuah tanda tangan yang dilakukan menggunakan cara mengkonversi *header* dan *payload* ke bentuk basis 64, menggabungkan keduanya dengan *delimiter* titik, lalu dilakukan *hash* menggunakan algoritma *hash* yang diminta. Proses selengkapnya dapat dilihat di bawah:

```
h_b64       = base64(header)
p_b64       = base64(payload)
data        = h_64 + "." + p_64
signature   = Hash(data)
```

4. Pembentukan kunci, misalkan pada contoh ini hasil basis 64 dari *header*, *payload*, dan *signature* pada contoh sebelumnya sebagai berikut.

maka kunci yang terbentuk berasal dari gabungan ketiganya dengan *delimiter* titik, misal pada contoh di bawah.

```
jwt_token = "abcABC.defDEF.1X2Y3Z"
```

5. Verifikasi JWT *token*, proses verifikasi dilakukan dengan mengonstruksi ulang proses 1 sampai 4 lalu menyocokkannya dengan *hash* yang dikirim. *Hash* yang cocok akan dikenali sebagai permintaan yang terpercaya, sedangkan yang tidak cocok akan ditolak.

F. API

API (*Application Programming Interface*) adalah suatu metode komunikasi *server* dengan *client* yang bervariasi (*multi platform*) dengan asal *request* yang bebas (tidak harus berasal dari domain server tersebut) dapat berupa *request anonymous*. API dibuat dapat untuk hampir segala bentuk komunikasi platform baik dari web, sistem operasi, basis data, maupun perangkat keras. Bentuk spesifikasi API dapat bervariasi, namun pada umumnya memberikan penjelasan berupa penggunaan dan implementasi.

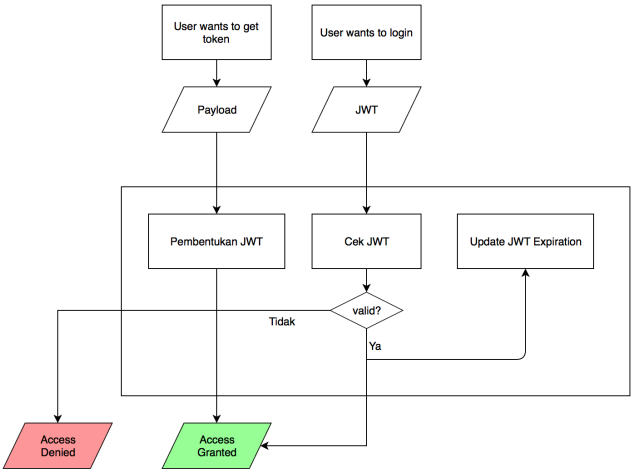
Tujuan utama dari API adalah mempermudah pekerjaan komunikasi *client-server*. API mempermudah proses otomasi yang dibuat menggunakan *script*/program tanpa harus melakukan klik pada antarmuka. Misal sebuah kebutuhan ketika programmer ingin melakukan pengiriman email secara berkala setiap hari tanpa harus membuka dan melakukan klik pada aplikasi email miliknya, maka cara yang mudah untuk dilakukan adalah dengan membuat sebuah program/*scipt* yang mengeksekusi API email tersebut secara rutin setiap hari.

Request yang dilakukan oleh API biasanya berupa *request* anonim yang tidak dikenali oleh server. Bentuk anonim ini mengharuskan sebuah identitas dari *request* agar dikenali oleh server. Cara pengenalan dapat dilakukan dengan banyak cara misalnya menggunakan email dan *password*. Namun penggunaan email dan *password* langsung pada *body request* dapat berbahaya karena dapat disadap (*sniffing*) oleh jaringan. Solusi lainnya adalah menggunakan JWT.

III. Rancangan dan Implementasi Sistem

Tujuan perancangan sistem untuk melakukan simulasi autentikasi mendapatkan JWT dan melakukan akses *resource* pada server menggunakan API REST. Rancangan yang dilakukan berupa membuat modul JWT dan modul Autentikasi beserta server untuk testing. Skenario pengujian berupa menguji kebenaran program mengenali pengguna menggunakan JWT.

Alur kerja *server* dan *client* digambarkan sebagai berikut.



Gambar 1. Arsitektur REST API dengan autentikasi JWT

Terdapat dua kasus utama pada sistem ini, yaitu saat pengguna meminta jwt baru dan ketika pengguna ingin *login* (mengakses *resources*) pada server tersebut. Pengguna yang melakukan *get token* akan disimpan sebagai pengguna yang teridentifikasi dan mendapatkan akses *resources* pada server, dan pengguna yang memberikan JWT yang valid akan mendapat akses juga.

Implementasi server dilakukan menggunakan pustaka Flask pada bahasa pemrograman Python. Pengujian dilakukan menggunakan Postman dengan melakukan REST API *request* sesuai skenario yang akan ditentukan berikutnya.

A. *Modul JWT*

Modul ini terdiri dari empat dari lima proses utama seperti didefenisikan pada Landasan Teori bagian JSON *Web Token*. Membuat *header*, *payload*, *signature*, dan pembentukan jwt. Modul ini digunakan oleh modul Autentikasi untuk mengambil jwt *token* dari *payload* identifikasi pengguna.

B. *Modul Autentikasi*

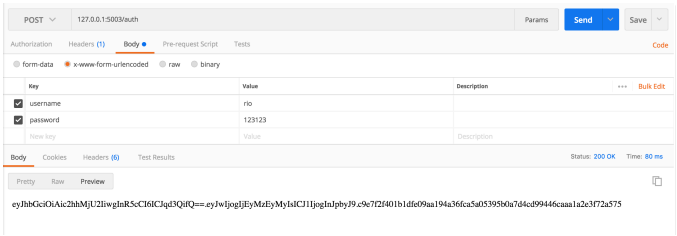
Modul ini berisi tahap kelima dari proses utama seperti didefenisikan pada Landasan Teori bagian JSON *Web* Token. Pada modul ini diimplementasikan fungsi pengguna mengambil *token* baru dan mengidentifikasi *token* yang telah dimilikinya. Terdapat modifikasi pada modul ini yaitu *token* diberikan batas waktu kadaluarsa sebesar 10 detik. Setelah 10 detik, *token* yang dimiliki oleh pengguna akan kadaluarsa. Pembaharuan waktu kadaluarsa *token* dilakukan setiap kali pengguna tersebut melakukan akses *resources* dengan *token* valid tersebut.

IV. EKSPERIMEN

Eksperimen dilakukan dengan tiga skenario yaitu: 1) pengambilan *token* baru, 2) akses *resources* dilakukan dengan pengecekan validasi *token* baru, dan 3) pengecekan *token* validasi ketika melewati waktu kadaluarsa.

A. *Pengambilan Token Baru*

Pengujian pengambilan *token* baru dilakukan menggunakan *payload* berupa *username* dan *password*.



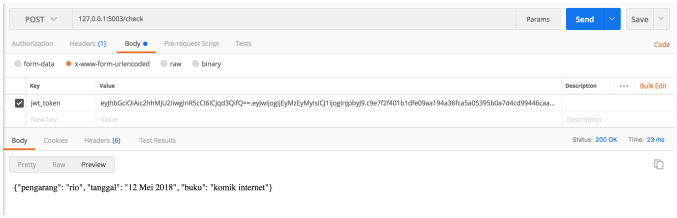
Gambar 2. Contoh autentikasi JWT

menggunakan *username* "rio" dan *password* "123123" dihasilkan JWT berupa.

eyJhbGciOiAic2hhMjU2IiwgInR5cCI6ICJqd3QifQ==.eyJwIjogIjEyMzEyMyIsICJ1IjogInJpbyJ9.c9e7f2f401b1dfe09aa194a36fca5a05395b0a7d4cd99446caaa1a2e3f72a575

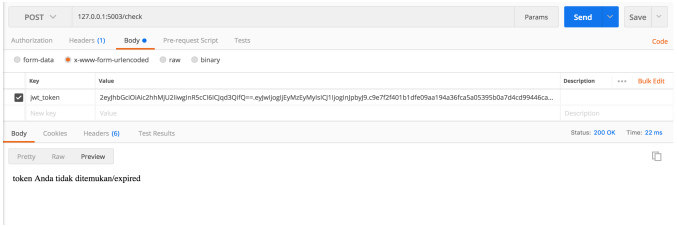
B. *Pengecekan Token*

Pengecekan *token* dilakukan dengan mencoba mengakses *resource* (pada kasus ini berupa data buku) dengan *passing* data berupa jwt *token* yang didapatkan.



Gambar 3. Contoh akses dengan JWT valid

Token valid seperti pada Gambar 3. menghasilkan *resource* yang di-*request*, sedangkan *token* yang tidak valid akan ditolak seperti pada Gambar 4.



Gambar 4. Contoh akses dengan JWT tidak valid

C. *Pengecekan Token Expired*

Token yang sudah *expired* akan menghasilkan keluaran yang sama dengan keluaran pada Gambar 4. *Token* yang kehabisan masa kadaluarsa (*expired*) harus di-*request* kembali lagi menggunakan *payload* data autentikasi yang valid. Setiap kali *request* dilakukan dengan jwt yang valid, masa kadaluarsa akan diperbaharui secara otomatis.

D. *Analisis Implementasi*

Proses implementasi pembangkit JWT pada REST API tidaklah rumit, beberapa kendala yang dihadapi adalah proses implementasi fungsi validasi jwt. Proses ini memiliki banyak opsi diantaranya dengan menyimpan jwt pada data server, atau melakukan *generate* ulang seluruh *token*, atau menyimpan beberapa bagian saja mis. hanya *header*, *payload*, dan lainnya. Pada skala ekperimen proses yang dilakukan bisa dengan melakukan *generate* ulang seluruh *token* setiap kali *request* oleh pengguna, tetapi untuk skala besar cara ini mungkin terlalu memakan *resource* yang besar, dibutuhkan cara-cara yang lebih efektif dalam mengelola jwt dan validasinya.

V. KESIMPULAN DAN SARAN

A. *Kesimpulan*

Implementasi JWT pada API (REST API) dapat menyelesaikan masalah CORS untuk mengidentifikasi pengguna yang diinginkan melakukan akses terhadap server. Beberapa proses optimasi mungkin dibutuhkan ketika melakukan implementasi pada server skala besar untuk meminimalkan proses komputasi validasi *token*.

B. *Saran*

Proses implementasi validasi *token* yang dilakukan pada makalah ini dengan mengirimkan jwt pada *body request*, untuk menghasilkan *request* yang lebih elegan pada implementasi server sebenarnya, jwt sebaiknya dikirim pada *header* bukan *body request*.

REFERENCES

[1] ElGamal, T. 1985. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.

[2] Munir, Rinaldi. 2018. Slide Kuliah IF4020 Kriptografi: Algoritma Kriptografi Modern.

[3] Munir, Rinaldi. 2018. Slide Kuliah IF4020 Kriptografi: Fungsi Hash.

[4] Clarke, Steven. 2004. Measuring API Usability, Dr. Dobb's.

[5] de Figueiredo, Luiz Henrique; Ierusalimschy, Roberto; Filho, Waldemar Celes. 2016. The design and implementation of a language for extending applications.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Mei 2018

Rio Chandra Rajagukguk