IMPLEMENTASI AUTENTIKASI BERBASIS TOKEN MENGGUNAKAN PLATFORM-AGNOSTIC SECURITY TOKENS (PASETO) SEBAGAI MEKANISME AUTENTIKANSI RESTFUL API

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Nico F. Sitorus
NIM: 145150201111096

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2020

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

PENGESAHAN

IMPLEMENTASI AUTENTIKASI BERBASIS TOKEN MENGGUNAKAN PLATFORM-AGNOSTIC SECURITY TOKENS (PASETO) SEBAGAI MEKANISME AUTENTIKANSI RESTFUL API

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Nico F. Sitorus
NIM: 145150201111096

Skripsi ini telah diuji dan dinyatakan lulus pada
1 Juli 2020
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing 2
2820nfs	
Ari Kusyanti, S.T., M.Sc.	Adhitya Bhawiyuga, S.Kom., M.Sc.
NIP: 19831228 201803 2 002	NIP: 19890720 201803 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

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

Repository Universitas Brawijaya

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 referensi.

Apabila ternyata di dalam 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, 13 Juni 2020

Nico F. Sitorus

NIM: 145150201111096

PRAKATA

Puji syukur Tuhan Yesus Kristus, oleh karena anugerah dan kasih setia-Nya yang besar akhirnya penulis dapat menyelesaikan penulisan skripsi dengan judul: "Implementasi Autentikasi Berbasis Token Menggunakan Platform-Agnostic Security Tokens (PASETO) Sebagai Mekanisme Autentikansi RESTful API". Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan karena segala keterbatasan yang ada. Untuk itu demi sempurnanya skripsi ini, penulis membutuhkan dukungan dan sumbangsih pikiran yang berupa kritik dan saran yang bersifat membangun.

Skripsi ini penulis persembahkan kepada kedua orang tua (Dompak Sitorus, S.Pd., M.M. dan Suarni Sirait, S.Pd.) yang telah tulus memberikan kasih sayang, doa, perhatian, dukungan moral dan materil yang telah diberikan selama ini. Terima kasih telah meluangkan segenap waktunya untuk mengasuh, mendidik, membimbing, dan mengiringi perjalanan hidup penulis dengan dibarengi dukungan dan bantuan tiada henti agar penulis sukses dalam menggapai cita-cita.

Untuk teman-teman penulis Ivan Faturahman, Muhammad Alvian Syaputra, Moh. Abi Ramadhani, Muhammad Kevin A.R, Adhirajasa Raitsany, Eldyto Puspa Laksana, Annureza Akbar, terima kasih telah membantu penulis dalam penyelesaian skripsi ini. Terima kasih telah membantu penulis dalam dukungan moral dan pemberian saran dan kritik selama perkuliahan dan penulisan skripsi.

Penyusunan skripsi ini tidak lepas dari bantuan berbagai pihak, karena itu penulis menyampaikan ucapan terima kasih kepada:

1. Ibu Ari Kusyanti, S.T., M.Sc. sebagai dosen pembimbing 1.
2. Bapak Adhitya Bhawiyuga, S.Kom., M.Sc., sebagai dosen pembimbing 2.
3. Untuk MTG Arena dan Dota 2 yang telah menemani di saat masa-masa kelam.
4. Untuk Keluarga besar penulis tanpa terkecuali yang telah mendukung dan mendoakan selama ini.

Kiranya skripsi ini dapat memberikan manfaat dan masukan bagi pembaca.

Terima Kasih.

Malang, 13 Juni 2020

Penulis

nicositorus@student.ub.ac.id

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

Repository Universitas Brawijaya

ABSTRAK

Nico F. Sitorus, Implementasi Autentikasi Berbasis Token Menggunakan Platform-Agnostic Security Tokens (PASETO) Sebagai Mekanisme Autentikansi RESTful API

Pembimbing: Ari Kusyanti dan Adhitya Bhawiyuga

RESTful API merupakan standar arsitektur komunikasi client-server untuk layanan web, namun rendah dari segi keamanan. Pengamanan data RESTful API menggunakan autentikasi untuk membatasi akses pada API dengan tidak menyimpan *session state,* sehingga diperlukan autentikasi berbasis token yang dikirim melalui *header request* seperti JSON Web Token (JWT). Token JWT dapat dipalsukan dengan mengeksploitasi algoritme *none*, memanfaatkan mekanisme JWT yang selalu percaya pada header berisi algoritma yang digunakan, dan penggunaan algoritma HMAC yang sudah dapat dieksploitasi. Pemalsuan token tersebut dapat diatasi dengan tidak menggunakan algoritma none dan mengubah mekanisme JWT. Mekanisme JWT dapat diganti dengan standar yang hanya menyediakan versi dan tujuan protokol kriptografi yang digunakan dan selalu membandingkan versi dan tujuan protokol kriptografi yang digunakan dengan header yang dikirim seperti PASETO. Algoritma HMAC juga dapat diganti dengan algoritma BLAKE2b dengan kekuatan setara dengan SHA-3, namun lebih cepat dari SHA-1 dan MD5. PASETO juga menggunakan algoritma XChaCha20-Poly1305 untuk enkripsi dan algoritma Ed25519 untuk *signature*. PASETO dapat diimplementasikan menggunakan python, menggunakan *wrapper* pada *library* Libsodium agar dapat digunakan pada Python, dan childprocess agar dapat digunakan pada nodeJS.

Kata kunci: *RESTful, JSON Web Token, Platform-Agnostic Security Tokens, XChaCha20-Poly1305, Ed25519*

Repository Universitas Brawijaya

ABSTRACT

Nico F. Sitorus, Implementation of Token Based Authentication Using Agnostic Security Tokens (PASETO) as an implementation of RESTful API Authentication

Supervisors: Ari Kusyanti dan Adhitya Bhawiyuga

Abstract

RESTful *API is a client-server communication architecture standard for web services, but low in terms of security. RESTful API data security uses authentication to limit access to the* API *by not saving the session state, so that token-based authentication is required which is sent via a request header such as* JSON Web Token *(*JWT*). JWT tokens can be forged by exploiting None algorithm, utilizing JWT mechanism which always believes in token's header that contains token's algorithm used, and using* HMAC *algorithm which was already to be exploited. Token forgery can be overcome by not using* None *algorithm and changing* JWT's *mechanism. JWT's mechanism can be replaced with a standard that only provides the version and purpose of the cryptographic protocol used and always compares the version and purpose of the cryptographic protocol used with the token's header had sent.* HMAC *algorithm can be replaced with* BLAKE2b *algorithm. Its power equivalent to* SHA-*3, but faster than* SHA-*1 and* MD5. PASETO *also uses XChaCha20-Poly1305 algorithm for encryption and* Ed25519 *algorithm for signatures.* PASETO *can be implemented using python, using a wrapper for* Libsodium *library so that it can be used in python, and childprocess with nodeJS.*

Keywords: *RESTful, JSON Web Token, Platform-Agnostic Security Tokens, XChaCha20-Poly1305, Ed25519*

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
PRAKATA iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL x
DAFTAR GAMBAR xi
DAFTAR LAMPIRAN xii
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Identifikasi Masalah 3
1.3 Rumusan masalah 3
1.4 Tujuan 3
1.5 Manfaat 4
1.6 Batasan masalah 4
1.7 Sistematika pembahasan 4
BAB 2 LANDASAN KEPUSTAKAAN 6
2.1 Kajian Pustaka 6
2.2 Dasar Teori 8
2.2.1 RESTful API 8
2.2.2 JSON Web Tokens (JWT) 9
2.2.3 Libsodium 10
2.2.4 Platform-Agnostic SEcurity Tokens (PASETO) 12
BAB 3 METODOLOGI 17
3.1 Identifikasi Masalah 18
3.2 Studi Literatur 22
3.3 Analisis Kebutuhan 23
3.3.1 Analisis Kebutuhan Perangkat Keras 23
3.3.2 Analisis Kebutuhan Perangkat Lunak 23

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

3.3.3 Identifikasi Aktor 24
3.3.4 Kebutuhan Fungsional 24
3.3.5 Kebutuhan Nonfungsional 25
3.4 Perancangan Sistem 25
3.4.1 Perancangan Arsistektur Sistem 25
3.4.2 Perancangan Server RESTful API 27
3.4.3 Perancangan Pengujian 37
3.5 Implementasi 38
3.6 Pengujian dan Analisis 38
3.7 Pengambilan Kesimpulan 38
BAB 4 IMPLEMENTASI 39
4.1 Gambaran Umum Sistem 39
4.2 Implementasi 39
4.2.1 Konfigurasi Aplikasi 39
4.2.2 Database 40
4.2.3 Server 40
4.2.4 Skema *User* 41
4.2.5 Autentikasi *Controller* 41
4.2.6 Pembuatan Token 47
4.2.7 Penguraian Token 48
4.2.8 PASETO 49
BAB 5 PENGUJIAN DAN ANALISIS 59
5.1 Pengujian Fungsional 59
5.1.1 Tujuan Pengujian 59
5.1.2 Prosedur Pengujian 59
5.1.3 Hasil dan Analisis 60
5.2 Pengujian Nonfungsional 64
5.2.1 Tujuan Pengujian 64
5.2.2 Prosedur Pengujian 64
5.2.3 Hasil dan Analisis 64
5.3 Pengujian Keamanan 65
5.3.1 Tujuan Pengujian 65

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

5.3.2 Prosedur Pengujian 65
5.3.3 Hasil dan Analisis 66
5.4 Pengujian Performa 70
5.4.1 Tujuan Pengujian 70
5.4.2 Prosedur Pengujian 70
5.4.3 Hasil dan Analisis 71
BAB 6 PENUTUP 75
6.1 Kesimpulan 75
6.2 Saran 75
DAFTAR PUSTAKA 76
LAMPIRAN A PERBANDINGAN KINERJA PASETO DAN JWT DALAM PEMBUATAN TOKEN 78
A.1 Pembuatan Token 78
A.2 Penguraian Token 80

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

DAFTAR TABEL

Tabel 2.1 Kajian Pustaka 7
Tabel 2.2 Fungsi GenericHash 11
Tabel 2.3 Tabel PAE 11
Tabel 3.1 Aktor 24
Tabel 5.1 Pengujian POST /api/auth/register 60
Tabel 5.2 Pengujian POST /api/auth/login 61
Tabel 5.3 Pengujian GET /api/auth/me 61
Tabel 5.4 Pengujian GET /api/auth/allData 62
Tabel 5.5 Pengujian GET /api/auth/logout 64
Tabel 5.6 *Hashing password* 64
Tabel 5.7 Base64 Enkode/Dekode 66
Tabel 5.8 Eksploitasi Algoritme None 67
Tabel 5.9 Eksploitasi Algoritme Simetris-Asimetris 68
Tabel 5.10 Waktu yang dibutuhkan untuk pembuatan token (ms) 71
Tabel 5.11 Waktu yang dibutuhkan untuk penguraian token (ms) 73
Tabel 6.1 Waktu Yang Dibutuhkan Untuk Pembuatan Token (ms) 78
Tabel 6.2 Waktu Yang Dibutuhkan Untuk Penguraian Token (ms) 80
Tabel 6.2 Waktu Yang Dibutuhkan Untuk Penguraian Token (ms) 82

DAFTAR GAMBAR

Gambar 3.1 Diagram Alir Penelitian 17
Gambar 3.2 Alur Sistem Keamanan RESTful API 18
Gambar 3.3 Alur Sistem Keamanan RESTful API Menggunakan JWT 19
Gambar 3.4 Verifikasi Token JWT 20
Gambar 3.5 Verifikasi Token Palsu 20
Gambar 3.6 Algoritme HMAC 21
Gambar 3.7 Algoritma RSA256 21
Gambar 3.8 Validitas Token Palsu 22
Gambar 3.9 Arsitektur Sistem 26
Gambar 3.10 Use Case Diagram Server 27
Gambar 3.11 Sequence Diagram Register 29
Gambar 3.12 Sequence Diagram Login 31
Gambar 3.13 Sequnce Diagram Rute Me 34
Gambar 3.14 Sequence Diagram allData 36
Gambar 3.15 Sequence Diagram Logout 37
Gambar 5.1 Token v2.public 68
Gambar 5.2 Enkode Payload 68
Gambar 5.3 Verifikasi Token 68
Gambar 5.4 Token v2.public 69
Gambar 5.5 Enkode Payload 70
Gambar 5.6 Enkripsi 70
Gambar 5.7 Verifikasi token 70

DAFTAR LAMPIRAN

LAMPIRAN A PERBANDINGAN KINERJA PASETO DAN JWT DALAM PEMBUATAN TOKEN 78
A.1 Pembuatan Token 78
A.2 Penguraian Token 80

BAB 1 PENDAHULUAN

1.1 Latar belakang

Application Program Interface (API) secara umum adalah ekspresi abstrak dari syarat dan tindakan sebuah sistem, baik perintah, fungsi dan protokol yang dapat digunakan untuk membangun dan berinteraksi dengan sistem (Emery, 2005). Untuk membangun sebuah API yang baik maka diperlukan penamaan *path* API yang baik, memiliki respon balik dari API yang dikenali, memiliki pesan *error* ketika *error* terjadi, mudah diperbaharui, dan lain-lain (Bloch, 2006). Untuk itu, maka dibutuhkan sebuah standar arsitektur komunikasi seperti *Representational State Transfer* (REST).

REST adalah standar arsitektur atau gaya komunikasi yang memiliki penamaan *path* API yang teratur. REST merupakan standar arsitektur komunikasi berbasis *client-server* yang sering digunakan dalam pengembangan layanan berbasis web. Umumnya menggunakan *Hypertext Transfer Protocol* (HTTP) sebagai protokol untuk komunikasi data, sehingga dapat digunakan untuk membantu mengenali respon balik dari API melalui HTTP *status code* (Fielding, 2000). Arsitektur REST memiliki 2 tipe, yaitu REST server untuk menyediakan *data source* dan REST c*lient* untuk mengakses dan menampilkan *data source*. Tiap *data source* akan diidentifikasi menggunakan *Uniform Resource Identifier* (URI) *link*. REST menggunakan beberapa format data seperti teks, *JavaScript Object Notation* (JSON), dan *Extensible Markup Language* (XML). Metode HTTP umum yang digunakan REST untuk berinteraksi dengan *data source* berupa GET, PUT, DELETE, POST (Richardson, 2007). Untuk implementasinya maka dapat menggunakan RESTful API yang mengikuti pola dan protokol dari REST.

Namun RESTful API sangat rendah dalam segi keamanan (Kumari, 2015). Untuk pengamanan data, biasanya RESTful API menggunakan autentikasi untuk membatasi akses pada API. Namun API tidak menyimpan *session state* untuk menyimpan *auth* atau *stateless,* sehingga diperlukan autentikasi berbasis token yang dikirim melalui *header request* seperti JSON Web Token (JWT) (Satria, 2018). JWT pada umumnya memiliki 3 bagian, yaitu *header*, *payload*, dan *signature*. *Header* mengidentifikasi algoritme mana yang akan digunakan untuk menghasilkan *signature,* seperti HS256 mengindikasikan bahwa token ditandatangani menggunakan HMAC-SHA256. *Payload* berisi informasi yang dikirimkan, sedangkan *signature* adalah hasil kalkulasi *header* dan *payload* yang dienkode menggunakan Base64url kemudian ditandatangani menggunakan algoritme yang telah tertera pada header dan *secret key* yang telah ditentukan (Bradley, 2015). Mekanisme verifikasi token yang digunakan oleh JWT cukup janggal karena untuk menentukan algoritme yang digunakan untuk menghasilkan *signature* dapat ditemukan pada *header*. Akan tetapi token tersebut belum divalidasi, yang artinya *header* yang sedang diterima juga belum divalidasi. Sehingga dapat membahayakan karena dapat mengizinkan penyerang untuk memilih metode yang digunakan untuk verifikasi *signature* (Arciszewski, 2017).

Selain itu JWT sendiri memiliki banyak kelemahan dalam arsitekturnya sendiri karena menggunakan *library* yang rentan (McLean, 2015). Selain itu data hanya dienkode menggunakan Base64-encode yang dapat didekode dengan mudah, contohnya menggunakan web JWT sendiri (Arciszewski, 2017). Selain itu token yang digunakan JWT dapat dipalsukan dengan mengubah *signing algorithm* seperti mengubah algoritme yang terdapat pada *header* dengan spesifikasi *None*. Algoritme *None* dimaksudkan digunakan apabila integritas sebuah token sudah diverifikasi sehingga tidak perlu diverifikasi lagi untuk selanjutnya. Akan tetapi banyak *library* yang digunakan JWT memperlakukan token dengan *header None* sebagai token yang valid tanpa melalui proses verifikasi. Sehingga memungkinkan pihak tak berwenang dapat masuk ke dalam sistem dengan membuat token dengan *payload* apa saja menggunakan *header None* (Langkemper, 2016).

JWT juga mendefinisikan beberapa algoritme seperti RSA256 dan HS256 sebagai salah satu algoritme yang dapat digunakan pada sistemnya. RSA256 menggunakan *private key* untuk menandatangani token yang dikirim dan *public key* untuk verifikasi. Artinya hanya orang tertentu yang dapat menandatangani token, tetapi siapa pun dapat memverifikasi apakah token ditandatangani dengan benar. Sebagian besar *library* JWT menggunakan *verificationKey* untuk verifikasi token. Sedangkan pada sistem yang menggunakan HS256, *verificationKey* adalah *signingKey* rahasia server karena HS256 menggunakan kunci yang sama untuk verifikasi dan tanda tangan. Yang menjadi masalah jika terdapat pemalsu token yang menggunakan algoritme HS256 untuk menandatangani token namun menggunakan *public key* RSA256. Artinya pemalsu token seolah-olah menggunakan menggunakan RSA256 namun sebenarnya menggunakan HS256. Sehingga server akan berasumsi token ditandatangani dengan RSA, tetapi menerima token yang ditandatangani dengan HMAC. Sehingga *public key* akan dianggap sebagai *secret key* HS256. Hal ini mengizinkan penyerang yang memiliki akses pada *public key* menggunakannya untuk memalsukan token (Langkemper, 2016).

Oleh karena itu, dibutuhkan sebuah mekanisme autentikasi berbasis token yang *payload*-nya dienkripsi agar tidak dapat informasi dalam token tidak diketahui oleh pihak tak berwenang, perlu adanya tanda tangan agar token tidak dapat dimodifikasi dengan mudah, dan penggunaan algoritma yang rentan. Untuk autentikasi berbasis token dengan enkripsi pada *payload* dapat menggunakan Branca namun tidak memiliki fitur tanda tangan (Branca, 2017). Selain itu, autentikasi berbasis token yang dapat digunakan adalah PASETO yang dapat melakukan enkripsi dan tanda tangan(Arciszewski, 2017). Dibandingkan dengan JWT yang membebaskan pengguna untuk memilih algoritme apa saja yang digunakan, Platform-Agnostic Security Tokens (PASETO) lebih mengarahkan pengguna untuk menggunakan versi PASETO mana yang digunakan agar tidak membingungkan. Informasi PASETO akan diletakkan pada header yang akan memberitahukan informasi mengenai token dienkripsi atau hanya ditandatangani saja, algoritme yang digunakan, dan *library* digunakan dalam bentuk kode tertentu yang hanya diketahui oleh server PASETO saja. Sehingga tidak ada kebingungan, tebak-tebakan dan kesalahpahaman antara klien dan server. Hal ini juga

mencegah penyerang untuk mengetahui algoritme yang digunakan sehingga mengurangi terjadinya pemalsuan token seperti pada JWT. Akan tetapi PASETO hanya dapat melakukan enkripsi saja atau tanda tangan saja. Dari permasalahan di atas dan bukti yang diuraikan, maka penulis melakukan penelitian yang mengubah mekanisme JWT menjadi mekanisme PASETO sebagai mekanisme autentikasi RESTful API dengan memodifikasi mekanisme PASETO sehingga dapat melakukan enkripsi dan tanda tangan di versi yang sama.

1.2 Identifikasi Masalah

Berdasarkan dari latar belakang yang telah dijelaskan dapat diidentifikasi masalah yang akan digunakan dalam penelitian ini, yaitu:

1. Pemalsuan token pada JWT dapat dilakukan dengan mengeksploitasi algoritme *none*. Hal ini dapat diatasi dengan tidak menggunakan algoritma none.
2. Pemalsuan token pada JWT dapat dilakukan dengan memanfaatkan *signature* simetris dan asimetris. Hal ini dapat diatasi dengan mengubah mekanismenya. JWT yang selalu percaya pada header yang diterima dapat diubah dengan mengganti standar JWT dengan standar yang hanya menyediakan parameter versi protokol kriptografi yang digunakan dan parameter operasi yang akan digunakan seperti yang diterapkan pada PASETO. Dimana PASETO selalu mengecek versi protokol dan operasi yang digunakan dan membandingkan dengan header yang dikirim. Jika berbeda maka langsung ditolak.
3. Pemalsuan token juga dapat dilakukan dengan memanfaatkan *secretKey* yang digunakan oleh HMAC untuk menandatangani dan memverifikasi token. Dapat diatasi dengan menggunakan algoritma BLAKE2b. BLAKE2b merupakan fungsi *hash* yang memiliki kekuatan setara dengan SHA-3, namun lebih cepat dari SHA-1 dan MD5. Dimana HMAC merupakan algoritma generasi SHA1.

1.3 Rumusan masalah

Berdasarkan dari latar belakang yang telah dijelaskan dapat diambil rumusan masalah yang dapat dijadikan sebagai bahan penelitian skripsi sebagai berikut:

1. Bagaimana menerapkan mekanisme autentikasi PASETO ke dalam RESTful API?
2. Bagaimana perbandingan kinerja JWT dan PASETO sebagai mekanisme autentikasi pada RESTful API?

1.4 Tujuan

Tujuan dari penelitian "Implementasi Autentikasi Berbasis Token Menggunakan Platform-Agnostic Security Tokens (PASETO) Sebagai Mekanisme Autentikansi RESTful API" adalah :

1. Mengimplementasikan PASETO pada standar arsitektur komunikasi RESTful API

2. Mengukur kinerja penerapan JWT dan PASETO pada standar arsitektur komunikasi RESTful API

1.5 Manfaat

Manfaat penelitian ini adalah memberikan alternatif lain dalam pengamanan data khususnya pada standar komunikasi RESTful API untuk autentikasi *user*.

1.6 Batasan masalah

Batasan masalah dalam penelitian ini adalah

1. Fokus penelitian ini adalah autentikasi user pada RESTful API.
2. PASETO yang digunakan adalah mekanisme PASETO versi 2.
3. *Email* yang digunakan oleh klien untuk pendaftaran selalu unik

1.7 Sistematika pembahasan

Sistematika penulisan skripsi ini ditunjukkan untuk memberi gambaran dan uraian dari penelitian secara garis besar yang meliputi beberapa bab sebagai berikut:

BAB 1 Pendahuluan

Bab ini menguraikan tentang latar belakang mengapa pentingnya pengamanan pada RESTful API, buruknya mekanisme yang digunakan JWT, dan solusi yang ditawarkan penulis. Bab ini juga membahas rumusan masalah yang akan didasarkan pada latar belakang, tujuan, manfaat, batasan masalah, dan sistematika penulisan pada penelitian yang akan dilakukan.

BAB 2 Landasan Kepustakaan

Bab ini menguraikan tentang dasar-dasar teori dengan membandingkan penelitian sebelumnya dengan penelitian yang akan dilakukan. Bab ini juga membahas mengenai informasi yang berkaitan RESTful API, JSON Web Token, dan PASETO yang penulis gunakan sebagai acuan dan dasar dalam penelitian.

BAB 3 Metodologi

Bab ini menguraikan tentang medote penelitian yang akan digunakan untuk menyelesaikan permasalahan penelitian yang dilakukan secara sistematik meliputi studi literatur, perancangan sistem, tahapan rencana implementasi, rencana pengujian dan analisis, dana bagaimana pengambilan keputusan dilakukan.

BAB 4 Hasil

Bab ini menguraikan tentang laporan hasil penelitian yang dilaksanakan berdasarkan metode/teknik penelitian yang digunakan dan menyajikan data yang mendukung hasil tersebut.

BAB 5 Pembahasan

Bab ini menguraikan tentang pembahasan yang didapatkan dari data yang dihasilkan dari penelitian untuk menjawab semua masalah yang diajukan pada rumusan masalah.

BAB 6 Penutup

Bab ini menguraikan tentang kesimpulan yang dihasilkan dari penelitian yang diperoleh berdasarkan tujuan yang telah diuraikan sebelumnya dan saran mengenai pengembangan penelitian yang lebih lanjut

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini akan membahas mengenai teori-teori yang berkaitan dengan penelitian-penelitian sebelumnya dan dasar teori yang dibutuhkan untuk menyelesaikan penelitian ini. Kajian pustaka berisi jurnal, artikel, dan literatur mengenai mekanisme keamanan pada RESTful, JWT, dan PASETO.

2.1 Kajian Pustaka

Pada kajian pustaka akan dibahas mengenai penelitian-penelitian yang mendukung penyelesaian penelitian ini. Seperti yang dikemukakan Eliyani (2016), aplikasi SIKASIR berbasis RESTful *web service* membutuhkan mekanisme pengamanan yang dikembangkan menggunakan ASP.NET WEB MVC. Mekanisme pengamanan yang digunakan berbasis autentikasi dasar, autentikasi berbasis klaim, dan autentikasi berbasis token. Beberapa dekade belakangan ini, autentikasi berbasis *cookies* merupakan autentikasi paling mudah pada aplikasi berbasis web. Namun sangat rumit apabila diterapkan di *multi-platform* yang menggunakan autentikasi berbasis server seperti yang diterapkan oleh (*Security Assertion Markup Language* (SAML). Untuk itu aplikasi SIKASIR menggunakan JWT karena formatnya yang pendek.

Lengkemper (2016) mengemukakan bahwa JWT memiliki kelemahan pada *library* yang digunakan oleh algoritmenya. JWT rentan terhadap penggunaan algoritme None yang mengabaikan signature yang diberikan. Selain itu JWT juga dapat dimanipulasi menggunakan algoritme simetris dan asimetris dengan menggunakan kunci *public* dan *private*. JWT yang menggunakan algoritme HS256 juga sudah dapat dipecahkan menggunakan alat tertentu sehingga token dapat dipalsukan. Hal ini dapat diatasi dengan tidak menyimpan data sensitif pada token yang akan dikirim. Selain itu juga dapat menggunakan kunci yang kuat sehingga alat yang digunakan untuk memecahkan kunci membutuhkan waktu yang lama.

Berbeda dengan Lengkemper, Satria (2018) menyatakan bahwa untuk JWT dapat diperbaiki dengan menambahkan algoritma tambahan seperti Blake2s pada konfigurasi *JSON Web Algorithm* (JWA). Blake2s adalah algoritme *hashing* dan dapat digunakan sebagai *message authentication code* (MAC) yang tahan terhadap serangan *herding*, *long-message second preimages* dan *length extension*. JWT dengan Blake2s digunakan untuk mengamankan API berbasis RESTful API. Akan tetapi selama JWT masih menggunakan mekanisme yang sama dalam penggunaan header, maka JWT masih dapat dieksploitasi.

Arciszewski (2017) menyatakan bahwa penggunaan JWT seharusnya dihindari dikarenakan banyaknya masalah yang ditemukan pada *library* yang digunakan JWT. Ketika penyerang dapat memalsukan token dengan mudah, maka dapat dipertimbangkan bahwa protokol tersebut sudah rusak. Pemalsuan token yang memanfaatkan penggunaan algoritme None dan algoritme HS256 menggunakan *public key* RSA256 membuktikan bahwa JWT rusak bukan karena *implementation bug,* melainkan standarnya yang buruk. Standar JWT menyediakan banyak ruang

terjadinya kesalahan implementasi khususnya ketika enkripsi asimetris terlibat. Dibanding menggunakan JWT, penelitian Arciszewski menyarankan mengganti standar JWT dengan standar yang hanya menyediakan parameter versi protokol kriptografi yang digunakan dan parameter operasi yang akan digunakan seperti yang diterapkan pada PASETO (Arciszewski, 2018). Akan tetapi PASETO masih memilki kelemahan dimana, token hanya mampu dienkripsi saja atau ditandatangani saja, tanpa bias melakukan keduanya dalam satu token yang sama.

Tabel 2.1 Kajian Pustaka

No.	Nama Peneliti, Tahun, Judul	Persamaan	Perbedaan	
			Penelitian terdahulu	Rencana Penelitian
1.	Muhammad K, Eliyani, 2016, *Token-Based Authentication Using JSON Web Token on* SIKASIR RESTful Web Service	Mekanisme autentikasi pada RESTful	Menggunakan JWT sebagai mekanisme autentikasi pada RESTful Web Service	Menggunakan PASETO sebagai mekanisme autentikasi pada RESTful-API
2.	Sjoerd Langkemper, 2016, *Attacking JWT authentication*	JWT adalah mekanisme keamanan autentikasi yang bermasalah	JWT masih aman dipakai jika data yang dikirimkan bukanlah data yang sensitif atau menggunakan kunci yang kuat dalam pembentukan *signature*	JWT tidak aman dipakai dikarenkan *library* yang dipakai rentan dan mekanisme yang digunakan ambigu sehingga perlu dilakukan perubahan dari segi *library* yang digunakan dan mekanisme pengamanan yang digunakan.
3.	Bagus Satria, Ari Kusyanti, Widhi Yahya, 2018, Implementasi Algoritme Blake2s pada JSON Web Token (JWT)	Mekanisme autentikasi pada REST-API	Menggunakan JWT sebagai mekanisme autentikasi pada REST-API	Menggunakan PASETO sebagai mekanisme autentikasi pada RESTful-API

	sebagai Algoritme Hashinguntuk Mekanisme Autentikasi Layanan REST-API			
4.	Scott Arciszewski, 2017, *No Way, JOSE! Javascript Object Signing and Encryption is a Bad Standard That Everyone Should Avoid*	JWT adalah mekanisme keamanan autentikasi yang bermasalah	Menggunakan *library* Libsodium untuk menyediakan fungsi *signature* seperti crypto_sign() atau crypto_auth() dan fungsi enkripsi seperti crypto_secretbox() atau crypto_box() untuk mengamankan *session*	Menggunakan PASETO yang berbasis libsodium yang menyediakan fungsi *signature* seperti algoritme Ed25519 dan enkripsi menggunakan algoritme *hashing* Blake2b dan XChaCha20-Poly1305.
5.	Scott Arciszewski, 2018, Paseto is a Secure Alternative to the JOSE Standards	Alternatif solusi untuk penggunaan JWT	Hanya diimplementasikan pada program PHP dengan protokol HTTP	Mengimplementasi-kan pada layanan RESTful API

2.2 Dasar Teori

2.2.1 RESTful API

API secara umum adalah ekspresi abstrak dari syarat dan tindakan sebuah sistem, baik perintah, fungsi dan protokol yang dapat digunakan untuk membangun dan berinteraksi dengan sistem (Emery, 2005). Untuk menggunakan API agar dapat berinteraksi dengan sistem, maka dibutuhkan sebuah standar arsitektur komunikasi. RESTful API didasarkan pada REST, gaya arsitektur dan pendekatan komunikasi web berbasis *client-server* yang sering digunakan dalam pengembangan layanan berbasis web. Umumnya menggunakan HTTP sebagai protokol untuk komunikasi data (Fielding, 2000)

Arsitektur RESTful memiliki 2 tipe, yaitu RESTful server untuk menyediakan *data source* dan RESTful *client* untuk mengakses dan menampilkan *data source*. Tiap

data source akan diidentiikasi menggunakan URI *link* dan server akan merespon dengan mengirimkan balik sebuah HTTP *response* sesuai permintaan klien. RESTful menggunakan beberapa format data seperti teks, JSON, dan XML. Komponen yang digunakan oleh HTTP *request*, yaitu:

1. HTTP *method* seperti GET, POST, PUT, DELETE sesuai dengan tugasnya masing-masing.
2. URI untuk mengetahui lokasi data di server.
3. HTTP *version*, seperti HTTP v1.1.
4. *Request Header*, berisi metadata seperti *authorization*, tipe klien dan lain-lain.
5. *Request Body*, data yang diberikan klien ke server seperti URI *params*.

Sedangkan komponen pada HTTP *response* adalah:

1. Response Code, status server terhadap request yang diminta seperti 200, 401, 404 dan lainnya.
2. HTTP *version.*
3. *Response header* yang berisi meta data seperti *content type*, *cache tag* dan yang lainnya.
4. *Repsonse Body* yang berisi data/*resource* yang diberikan oleh server baik berupa teks, JSON ataupun xml.

HTTP *method* umum yang digunakan RESTful untuk berinteraksi dengan *data source,* yaitu:

1. GET biasanya digunakan untuk membaca *resource* dari RESTful server.
2. POST biasanya digunakan untuk membuat *resource* baru di RESTful server.
3. PUT biasanya digunakan untuk memperbaharui *resource* di RESTful server.
4. DELETE biasanya digunakan untuk menghapus *resource* dari RESTful server.
5. OPTIONS, digunakan untuk mendapatkan operasi yang didukung pada *resource* dari server RESTful.

2.2.2 JSON Web Tokens (JWT)

JWT adalah representasi dari standar format yang ringkas yang penggunaannya ditujankan pada lingkungan terbatas seperti HTTP *Authorization headers* dan *URI query parameters*. Objek JWT akan dienkode dan ditransmisikan sebagai klaim berbentuk *JavaScript Object Notation* (JSON). Dimana klaim ini akan digunakan sebagai payload dari struktur *JSON Web Signature* (JWS) atau bisa digunakan sebagai *plaintext* dari struktur *JSON Web Encryption* (JWE). JSON akan dapat dilindungi dengan tanda tangan digital seperti *Message Authentication Code* (MAC) atau dienkripsi (Jones, 2015).

Struktur dasar JWT terdiri atas 3 bagian, yaitu:

1. ***Header***

Header terdiri dari 2 bagian yaitu algoritma yang digunakan seperti HS256 dan tipe yang digunakan seperti JWT.

2. ***Payload***

Payload berisi infomasi atau data yang dikirimkan atau disimpan dalam JWT atau sering disebut klaim.

3. ***Verify Signature***

Verify Signature adalah hasil dari proses *hashing* atau gabungan dari isi enkode *header* dan *payload* lalu ditambahkan kode rahasianya.

2.2.3 Libsodium

Libsodium merupakan salah satu cabang implementasi dari *library* NaCl. Libsodium merupakan *library* Sodium modern yang mudah digunakan untuk enkripsi, dekripsi, *signature*, *password hashing*, dan lain-lain (Libsodium, 2013). Libsodium bersifat *portable*, dapat diinstal dan kompatibel dengan *library* NaCl — library perangkat lunak untuk komunikasi jaringan, enkripsi, dekripsi, *signature,* dan lain-lain — dengan API yang cocok. Libsodium juga merupakan *library* yang multiplatform dan multi bahasa pemrograman. Sehingga dapat digunakan di beberapa varian *compiler* dan *operating system,* seperti Windows, Linux, iOS, dan Android.

Mirip dengan NaCL, Libsodium tidak menawarkan banyak fitur. Libsodium lebih membatasi fitur dan mengurangi fleksibilitas pengguna untuk memasangkan fitur atau algoritma satu dengan yang lainnya. Hal ini dilakukan agar mengurangi kesalahan pemasangan fitur yang tidak cocok atau menambah *vulnerability* sebuah fitur (NaCl, 2008). Libsodium mengambil pilihan yang fleksibel dan hanya mengimplementasikan fitur-fitur yang tepat.

2.2.3.1 *Nonce*

Nonce seperti istilah pada namanya — a *number used once* — adalah sekumpulan angka acak atau *pseudo-random number* dan sering mengandung *timestamp* di dalamnya yang hanya digunakan sekali saja (Rogway, 2004). *Nonce* sering kali digunakan dalam beberapa skenario kriptografi seperti autentikasi HTTP *digest,* pencegahan *replay attack* — penyadapan diiringi pengulangan sebuah transmisi data yang sah (Libsodium, 2013), hingga dianggap transmisi normal — dalam jaringan, hal yang dicari oleh penambang Bitcoin, inisialisasi vektor (IV), dan untuk kepentingan enkripsi dalam pembuatan *secret* dan *public key.*

2.2.3.2 BLAKE2b

Algoritme BLAKE2b merupakan algoritme yang digunakan oleh libsodium untuk fungsi *hash* crypto_generichash_* (Libsodium, 2013), sesuai standar RFC7693 (Saarinen & Aumasson, 2015). BLAKE2b merupakan fungsi *hash* yang memiliki kekuatan setara dengan SHA-3, namun lebih cepat dari SHA-1 dan MD5 (Libsodium, 2013). BLAKE2b dapat menahan serangan *hash length extension*

attacks, tidak seperti MD5, SHA-1, dan SHA-256. BLAKE2b digunakan untuk mengecek integritas sebuah *file* dan membuat identifikasi unik untuk indeks *arbitrary data* yang panjang, sehingga cocok digunakan untuk data berjenis JSON.

Fungsi crypto_generichash() seperti pada persamaan merupakan fungsi yang menempat sidik jari pada sebuah pesan "*in*" dengan panjang tertentu "*inlen*" dalam *bytes* untuk menghasilkan *output* "*out*" dengan ukuran tertentu menggunakan kunci tertentu. Ukuran *output* dapat diatur panjang datanya sesuai keinginan dengan ukuran minimal rekomendasi diatur pada crypto_generichash_BYTES. Ukuran *output* akan bervariasi diantara crypto_generichash_BYTES_MAX dan crypto_generichash_BYTES_MIN. Sehingga secara praktik, tidak mungkin ada dua *output* berbeda menghasilkan sidik jari yang sama. Jika kunci diisi dengan *NULL* dan Panjang kunci diisi dengan 0 maka fungsi *hash* BLAKE2b akan berubah menjadi MD5 atau SHA-1.

Tabel 2.2 Fungsi GenericHash

Algoritme 1: Fungsi generichash	
1	int crypto_generichash(unsigned char *out, size_t outlen,
2	const unsigned char *in, unsigned long long inlen,
3	const unsigned char *key, size_t keylen);

2.2.3.3 *Pre-Authentication Encoding*

Pesan dengan *multi-part* seperti *header, payload, dan footer* dienkode dengan cara tertentu sebelum dilempar ke fungsi kriptografi disebut sebagai *Pre-Authentication Encoding* (PAE). PAE menerima sekumpulan *string* dalam bentuk *array* dalam *docblock* untuk menandai kunci *integer*. Sedangkan fungsi LE(64) bertugas untuk enkode 64-bit *unsigned integer* menjadi *little-endian binary string* untuk mendukung bahasa pemrograman yang tidak mendukung *unsigned integer,* sehingga *bit* paling besar harus dihapus. Sehingga bentuknya menjadi seperti tabel 2.3.

Tabel 2.3 Tabel PAE

PAE	Hasil
PAE([])	"\x00\x00\x00\x00\x00\x00\x00\x00"
PAE([''])	"\x01\x00\x00\x00\x00\x00\x00\x00\ x00\x00\x00\x00\x00\x00\x00\x00
PAE(['test'])	"\x01\x00\x00\x00\x00\x00\x00\x00\ x04\x00\x00\x00\x00\x00\x00\x00test"
PAE()	Type Error

2.2.3.4 XChaCha20-Poly1305

XChaCha20-Poly1305 merupakan varian dari algoritma ChaCha20-Poly1305 yang diperkenalkan pada Libsodium 1.0.12. XChaCha20-Poly1305 ditujukan untuk operasi enkripsi pesan menggunakan *nonce* dan *key* untuk menjaga pesan tetap

rahasia. XChaCha20-Poly1305 biasanya digunakan secara umum menyimpan metadata khusus protokol seperti panjang dan enkode pada pesan yang dienkripsi.

ChaCha20-Poly1305 yang hanya mampu melakukan enkripsi pesan secara aman sampai sejumlah 2^{64} pesan dengan *key* yang sama tanpa adanya batas pada pesan yang dienkripsi (2^{64} untuk *tag* 128 bit). Sedangkan XChaCha20-Poly1305 mampu mengenkripsi pesan secara aman dengan jumlah tak terbatas dengan ukuran apa saja. XChaCha20-Poly1305 juga dapat menerima *nonce* acak untuk setiap pesan yang akan dienkripsi. Pada Libsodium, algoritme XChaCha20-Poly1305 dapat digunakan dengan menjalankan fungsi crypto_aead_xchacha20poly1305_ietf_*().

2.2.3.5 Ed25519

Ed25519 merupakan *key generator* yang digunakan oleh Libsodium berdasarkan *Edwards-Curve Digital Signature Algorithm* (EdDSA) sesuai standar RFC8032 (Libsodium, 2013). Ed25519 akan menghasilkan *secret key* dan *public key*. *Secret key* akan digunakan untuk menambahkan *signature* pada sejumlah pesan. Sedangkan *public key* akan digunakan oleh siapa saja untuk verifikasi *signature* yang sebelumnya ditambahkan pada sebuah pesan untuk mengetahui apakah signature dikeluarkan oleh pembuat *public key* itu sendiri.

Ed25519 dapat digunakan pada Libsodium dengan menjalankan fungsi crypto_sign dengan mode kombinasi dan mode terpisah. Pada mode kombinasi pesan ditandatangani dengan melampirkan gabungan *signature* dan pesan sebagai sm, pesan sebagai p dan kunci sebagai k. Sedangkan pada mode terpisah, pesan ditandatangani melampirkan gabungan *signature* dan pesan.

2.2.4 Platform-Agnostic SEcurity Tokens (PASETO)

PASETO adalah representasi aman secara kriptografi, kompak, dan *URL-safe* yang ditujukan untuk lingkungan yang dibatasi seperti *HTTP Cookies*, *HTTPAuthorization headers*, dan *URI query parameters*. Enkode PASETO ditransmisikan dalam bentuk objek JSON. Enkripsi pada PASETO dilakukan secara simetris. Sedangkan *signing* atau tanda tangan menggunakan kriptografi dengan *public key*.

Perbedaan utama PASETO dengan standar keluarga JOSE (JWS, JWE, JWK, JWA, dan JWT) adalah JOSE mengizinkan pengimplementasi dan pengguna untuk padu padan dengan algoritma kriptografi pilihan sendiri yang ditentukan pada *header alg* pada JWT, sedangkan PASETO sudah secara jelas menentukan versi protokol yang akan digunakan untuk pilihan mencegah konfigurasi tidak aman. Artinya informasi PASETO akan diletakkan pada *header* yang akan memberitahukan informasi mengenai token dienkripsi atau hanya ditandatangani saja, algoritme yang digunakan, dan *library* digunakan yang hanya diketahui oleh server PASETO saja. Sehingga tidak ada kebingungan, tebak-tebakan dan kesalahpahaman antara klien dan server, dan juga mencegah penyerang untuk mengetahui algoritme yang digunakan sehingga mengurangi terjadinya pemalsuan token seperti pada JWT.

2.2.4.1 Format Pesan PASETO

Format pesan PASETO mengandung 3 atau 4 bagian yaitu:

1. ***Version***

Version adalah sebuah *string* yang merepresentasikan versi protokol paseto yang digunakan, dengan nilai valid v1 dan v2.

2. ***Purpose***

Purpose adalah *string* pendek yang merepresentasikan tujuan dari token dengan nilai valid *local* untuk enkripsi autentikasi *shared-key* dan *public* untuk *public key digital signatures* tanpa enkripsi.

3. ***Payload***

Payload merupakan *string* yang mengandung data token. Jika *purpose* yang digunakan adalah *local* maka data token dienkripsi menggunakan *symmetric cipher*. Jika *purpose* yang digunakan adalah *public* maka data token tidak dienkripsi

4. ***Footer***

Setiap data opsional dapat ditambahkan ke footer. Data ini diautentikasi melalui inklusi dalam perhitungan *tag* autentikasi bersama dengan header dan payload. Footer tidak harus dienkripsi.

Version, *purpose*, *payload*, dan *footer* sebagai *optional* yang dipisahkan menggunakan tanda titik atau karakter ASCII yang direpresentasikan sebagai 2E. Format pesan PASETO adalah sebagai berikut:

Dengan opsional *footer*:

version.purpose.payload.footer

Tanpa opsional *footer*:

version.purpose.payload

Jika tidak ada *footer* maka tidak perlu menambahkan titik setelah *payload*.

2.2.4.2 *PASETO Version 1*

Paseto versi v1 merupakan *compability mode* yang hanya diterapkan apabila komputer tidak dapat menjalankan PASETO v2. Versi ini terdiri dari beberapa algortime kriptografi primitif yang tersedia pada sistem yang lawas. Versi *local* menggunakan enkripsi dengan algoritme AES-256-CTR dengan kunci sebesar 256 *bit* dan HMAC-SHA-384 menggunakan konstruksi *Encrypt-then-MAC.* Kunci yang digunakan untuk proses enkripsi dan autentikasi merupakan kunci dari hasil gabungan dari kunci yang asli dan setengah dari nonce yang difasilitasi oleh *HMAC-based Extract-and-Expand Key Derivation Function* (HKDF). Sedangkan untuk versi *public,* pesan harus ditandatangani menggunakan key sebesar 2048 *bit* menggunakan algoritme RSA Probabilistic Signature Scheme. Versi ini menyediakan autentikasi saja, namun tidak mencegah konten yang

ditandatangani dapat dibaca secara langsung oleh mereka yang tidak memiliki *public key* atau *private key*.

2.2.4.3 *PASETO Version 2 Local*

1. **Enkripsi**

Diberikan pesan m, kunci k, dan *footer* f, maka;

- Atur *header* h menjadi v2
- Atur *purpose* menjadi *local* sehingga h menjadi v2.local.
- Hasilkan 24 *bytes* acak menggunakan OS's CSPRING
- Kalkulasi pesan m menggunakan BLAKE2b dan 24 bit acak sebagai *key* sehingga menghasilkan keluaran dengan panjang 24 bit yang disebut *nonce* n.
- Gabungkan h, n, dan f secara berurutan menggunakan PAE sebagai *pre-authentication*.
- Kemudian enkripsi pesan menggunakan XChaCha20-Poly1305 dengan *interface* AED yang disediakan oleh Libsodium.
- Jika *footer* yang digunakan kosong maka hasil akhir menjadi h || b64(n || c).
- Jika *footer* yang digunakan tidak kosong maka hasil akhir menjadi h|| b64(n||c) || . || base64url(f)
- Simbol|| artinya menggabungkan

2. **Dekripsi**

Diberikan pesan m, kunci k, dan *footer* f, maka:

- Jika f tidak kosong, maka pengimplementasi dapat memverifikasi f yang ditambahkan pada token dengan mencocokkan dengan *string* f yang diharapkan menggunakan fungsi perbandingan *string* dengan waktu yang konstan
- Verifikasi pesan yang dimulai dengan v2.local, jika tidak *throw an exception.* Konstanta ini akan disebut sebagai h
- Kemudian dekode *payload* tanpa h, f, dan titik tambahan antara m dan f dari base64url menjadi biner mentah. Sehingga n ke 24 *byte* ke kiri dan c sisanya.
- Kemudian *dekripsi* c menggunakan XChaCha20-Poly1305, dan masukkan hasilnya ke dalam p
- Jika p kosong maka *throw an exception,* jika tidak maka hasil adalah p

Contoh format pesan PASETO adalah sebagai berikut:

```
v2.local.QAxIpVe-
ECVNI1z4xQbm_qQYomyT3h8FtV8bxkz8pBJWkT8f7Ht1OpbroPDEZUKop_vaglyp76
CzYy375cHmKCW8e1CCkV0Lflu4GTDyXMqQdpZMM1E60aoQW27gaRSvWBrR3IgbFIa0
AkuUFw.UGFyYWdvbiBJbml0aWF0aXZlIEVudGVycHJpc2Vz
```

Jika pesan di atas di enkode maka akan menghasilkan:

- *Version* : v2
- *Purpose* : *local (shared-key authenticated encryption)*
- *Payload* (*hex-encoded*) :
  400c48a557be10254d235cf8c506e6fea418a26c93de1f05b55f1bc64cfca4
  1256913f1fec7b653a96eba0f0c46542a8a7fbda825ca9efa0b3632dfbe5c1
  e62825bc7b5082915d0b7e5bb81930f25cca9076964c33513a39aa105b6e
  e06914af581ad1dc881b1486b4024b9417
- *Nonce* :
  400c48a557be10254d235cf8c506e6fea418a26c93de1f05
- *Authentication tag* : 6914af581ad1dc881b1486b4024b9417
- *Decrypted Payload* :

```
{
  "data": "this is a signed message",
  "exp": "2039-01-01T00:00:00+00:00"
}
```

- *Key* :
  707172737475767778797a7b7c7d7e7f808182838485868788898a8b8c8d
  8e8f
- *Footer* : Paragon Initiative Enterprises

2.2.4.4 *PASETO Version 2 Public*

1. ***Signing***

Diberikan pesan m, kunci rahasia Ed25519 sk, dan *footer* f (biasanya *string* kosong), maka;

- Atur *header* h menjadi v2
- Atur *purpose* menjadi *public* sehingga h menjadi v2.public.
- Gabungkan h, m, dan f menggunakan PAE (*pre-authentication encoding*), disebut sebagai m2
- Tandatangani m2 menggunakan sk Ed25519, disebut sebagai sig
- Jika *footer* yang digunakan kosong maka hasil akhir menjadi h || base64url(m || sig).
- Jika *footer* yang digunakan tidak kosong maka hasil akhir menjadi h|| base64url(m||sig) || . || base64url(f)
- Simbol|| artinya menggabungkan

2. **Verifikasi**

Diberikan pesan sm, kunci publik pk, dan footer f, maka:

- Jika f tidak kosong, maka pengimplementasi dapat memverifikasi f yang ditambahkan pada token dengan mencocokkan dengan *string* f yang diharapkan menggunakan fungsi perbandingan *string* dengan waktu yang konstan
- Verifikasi pesan yang dimulai dengan v2.public, jika tidak *throw an exception.* Konstanta ini akan disebut sebagai h
- Kemudian dekode *payload* sm tanpa h, f, dan titik tambahan antara m dan f dari base64url menjadi biner mentah. Sehingga s 64 *byte* ke kanan dan m sisanya.
- Gunakan Ed25519 untuk memverifikasi *signature* valid atau tidak
- Jika m valid maka kembalikan nilai m, jika tidak maka *throw an exception.*

Contoh format pesan PASETO adalah sebagai berikut:

```
v2.public.eyJleHAiOiIyMDM5LTAxLTAxVDAwOjAwOjAwKzAwOjAwIiwiZGF0YSI6
InRoaXMgaXMgYSBzaWduZWQgbWVzc2FnZSJ91gC7-
jCWsN3mv4uJaZxZp0btLJgcyVwL-svJD7f4IHyGteKe3HTLjHYTGHI1MtCqJ-
ESDLNoE7otkIzamFskCA
```

Jika pesan di atas di encode maka akan menghasilkan:

- *Version* : v2
- *Purpose* : *public (public-key digital signature)*
- *Signature* (*hex-encoded*) : d600bbfa3096b0dde6bf8b89699c59a746ed2c981cc95c0bfacbc90fb7f8207c86b5e29edc74cb8c761318723532d0aa27e1120cb36813ba2d908cda985b2408
- *Public Key* (*hex-encode*) : 11324397f535562178d53ff538e49d5a1622429705 56b4edd950c87c7d86648a

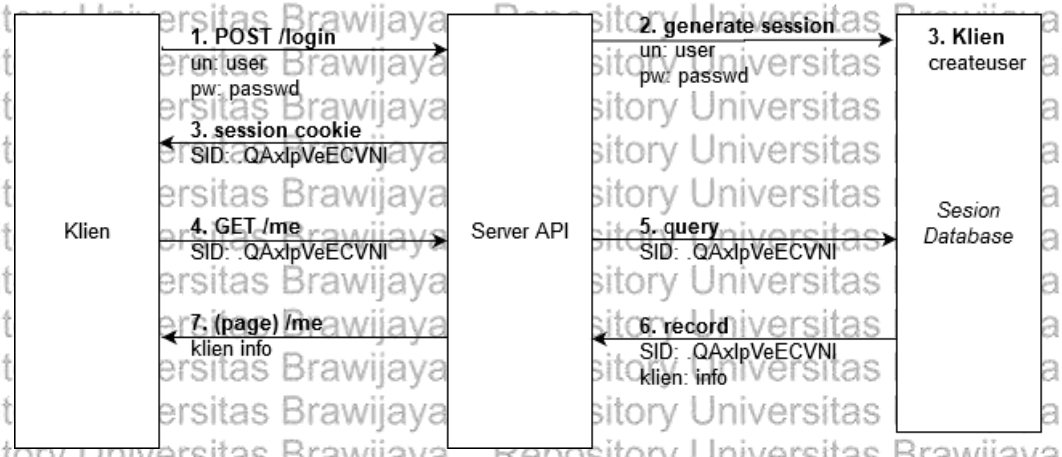
BAB 3 METODOLOGI

Bab ini membahas mengenai metode penelitian, waktu dan tempat penelitian, jadwal penelitian, alat dan bahan penelitian, langkah penelitian, diagram alur penelitian dan diagram sistem penelitian. Metode penelitian yang digunakan pada penelitian ini adalah metode penelitian implementatif. Tahapan yang digunakan "Implementasi Autentikasi Berbasis Token Menggunakan Platform-Agnostic Security Tokens (PASETO) Sebagai Mekanisme Autentikansi RESTful API" dapat dilihat pada diagram alir pada Gambar 3.1.

Gambar 3.1 Diagram Alir Penelitian

3.1 Identifikasi Masalah

Pada *session* tradisional dibutuhkan 3 entitas yaitu klien, server, dan *session database*. Misalkan terdapat aplikasi yang menyediakan fitur *login,* sehingga dibutuhkan session yang dapat dilihat pada gambar 3.2.

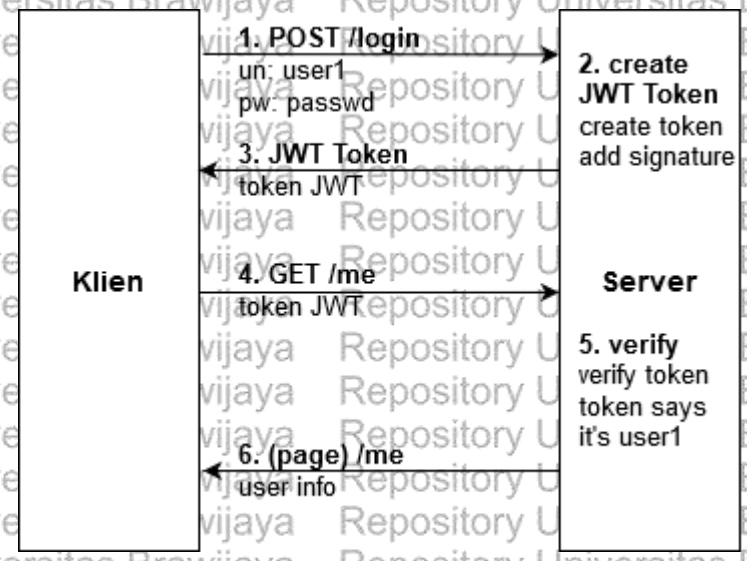


Gambar 3.2 Alur Sistem Keamanan RESTful API

Sederhananya klien mencoba *login* menggunakan *browser* dengan mengirimkan *username* dan *password*. Server akan mengecek apakah *username* dan *password* benar. Jika *username* dan *password* benar maka, server akan menghasilkan *session* berupa *string* acak yang terhubung dengan *user* yang kemudian disimpan pada *session database*. Kemudian server mengirimkan *session* berbentuk *cookie* pada *browser*. Pada saat klien ingin mengakses halaman tertentu, maka *browser* tidak perlu mengirimkan *username* dan *password* lagi, melainkan mengirimkan *session cookie* pada server. Karena server bersifat *stateless* maka server tidak mengenali klien, sehingga server tidak tahu *string* acak diasosiasikan kepada siapa, valid atau tidak, dan pernah dihasilkan server atau tidak. Sehingga server perlu mengecek ke *session database* untul mencari *string* acak tersebut pernah dibuat, valid, dan diasosiasikan kepada siapa. Jika valid, maka dapat diketahui siapa klien yang diasosiasikan dan server dapat mengirimkan halaman yang diminta oleh klien.

Jika RESTful API menggunakan autentikasi berbasis JWT maka entitas yang dibutuhkan hanya klien dan server, sedangkan *session database* tidak dibutuhkan. Sederhananya klien mencoba *login* menggunakan *browser* dengan mengirimkan *username* dan *password*. Server akan mengecek apakah *username* dan *password* benar. Jika *username* dan *password* benar maka, server akan membuat JWT Token *string* berisi *header* berupa algoritme yang digunakan, *payload* berupa informasi klien yang dikirimkan, dan *signature*. Kemudian server mengirimkan JWT token yang mengidentifikasi klien pada *browser*. Pada saat klien ingin mengakses halaman tertentu, maka *browser* tidak perlu mengirimkan *username* dan *password* lagi, melainkan mengirimkan JWT token menggunakan *header* atau *cookie* pada server. Sederhananya klien mencoba *login* menggunakan *browser* dengan mengirimkan *username* dan *password*. Server akan mengecek apakah *username* dan *password* benar. Jika *username* dan *password* benar maka, server akan membuat JWT Token *string* berisi *header* berupa algoritme yang digunakan,

payload berupa informasi klien yang dikirimkan, dan *signature*. Kemudian server mengirimkan JWT token yang mengidentifikasi klien pada *browser*. Pada saat klien ingin mengakses halaman tertentu, maka *browser* tidak perlu mengirimkan *username* dan *password* lagi, melainkan mengirimkan JWT token menggunakan *header* atau *cookie* pada server. Alur sistem JWT dapat dilihat pada gambar 3.3.



Gambar 3.3 Alur Sistem Keamanan RESTful API Menggunakan JWT

Karena server bersifat *stateless* maka server tidak mengenali klien, sehingga server tidak tahu *string* acak diasosiasikan kepada siapa, valid atau tidak, dan pernah dihasilkan server atau tidak. Kali ini server tidak memerlukan pihak ketiga untuk mengecek token yang dikirim, karena token sudah berisi informasi yang dibutuhkan. Hal yang penting untuk diperiksa oleh JWT di dalam server adalah *signature*. Jika *signature* valid, maka server dapat mengirimkan halaman yang diminta oleh klien.

Alur sistem yang digunakan JWT terlihat lebih bagus karena desain yang lebih jelas dan sedikit entitas yang berinteraksi, namun terdapat beberapa kelemahan. seperti data yang digunakan bersifat publik. Artinya jika token didapatkan, maka informasi yang terdapat dalam token dapat dibaca dengan mudah. *Header* dan *payload* pada token dapat didekode menggunakan Base64Url-decode. Selain itu , dengan tidak adanya *token database,* maka seseorang dapat memalsukan token dan membuat *signature* yang valid, maka isi token dapat diubah. Dan sistem tidak dapat mengetahui apakah token yang digunakan dihasilkan oleh sistem itu sendiri atau pihak lain.

Pemalsuan token pada JWT dapat dilakukan dengan mengeksploitasi algoritme *none*. Algoritme *none* merupakan algoritme tambahan aneh pada JWT. Algoritme none dimaksudkan digunakan apabila integritas sebuah token sudah diverifikasi sehingga tidak perlu diverifikasi lagi untuk selanjutnya. Dengan menggunakan algoritme *none*, maka token tidak memerlukan *signature* sehingga dapat dipalsukan dengan mudah. Dengan melakukan dekode pada token seperti pada

gambar 3.4 maka didapatkan beberapa informasi seperti algoritme yang digunakan pada *header* dan *claim data* pada *payload*.

```
Valid JWT: Jwt\Token Object
(
    [header:Jwt\Token:private] => Jwt\Header Object
        (
            [data:Jwt\Header:private] => Array
                (
                    [typ] => JWT
                    [alg] => HS256
                )

        )

    [payload:Jwt\Token:private] => Jwt\Payload Object
        (
            [data:Jwt\Payload:private] => Array
                (
                    [iss] => http://demo.sjoerdlangkemper.nl/
                    [iat] => 1571693148
                    [exp] => 1571693268
                    [data] => Array
                        (
                            [hello] => world
                        )

                )

    )
```

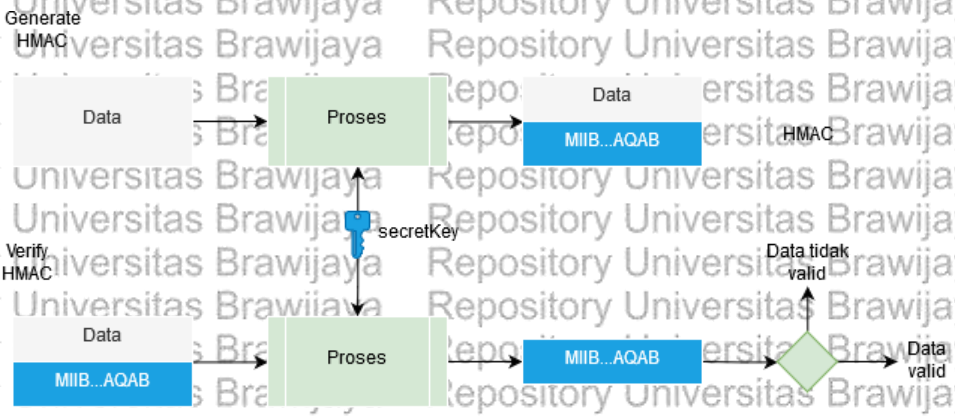
Gambar 3.4 Verifikasi Token JWT

Dengan informasi yang didapatkan, *token* dapat dipalsukan dengan mengubah algoritma yang digunakan menjadi *None*. Kemudian dekode kembali *header* dengan tipe algoritme none menggunakan base64Url. Setelah *header* dengan algoritme *None* didapatkan maka token dapat dipalsukan dengan mengganti *header* dan menghilangkan *signature*. Sehingga ketika diverifikasi token tetap valid seperti gambar pada 3.5

```
Valid JWT: Jwt\Token Object
(
    [header:Jwt\Token:private] => Jwt\Header Object
        (
            [data:Jwt\Header:private] => Array
                (
                    [typ] => JWT
                    [alg] => none
                )

        )

    [payload:Jwt\Token:private] => Jwt\Payload Object
        (
            [data:Jwt\Payload:private] => Array
                (
                    [iss] => http://demo.sjoerdlangkemper.nl/
                    [iat] => 1571693148
                    [exp] => 1571693268
                    [data] => Array
                        (
                            [hello] => world
                        )

                )

    )
```

Gambar 3.5 Verifikasi Token Palsu

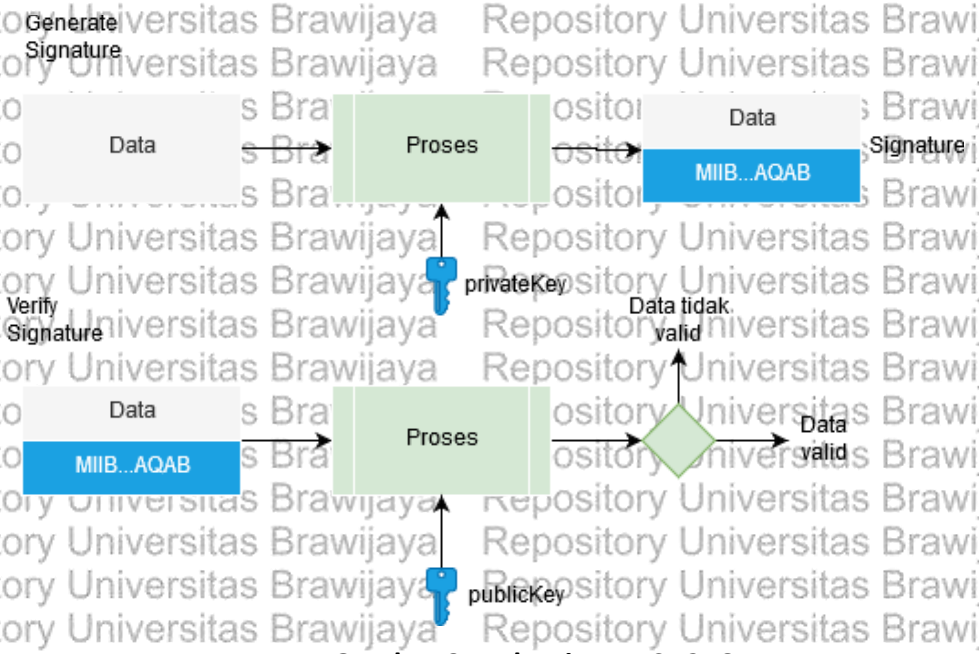
Pemalsuan token juga dapat dilakukan dengan memanfaatkan *signature* simetris dan asimetris. Pada JWT, terdapat algoritma simetris seperti *Hash-based Message Authentication Code* (HMAC/HS256) yang menggunakan *secret key* untuk menandatangani. Sedangkan untuk memverifikasi sebuah data, HMAC juga menggunakan *secret key* yang sama seperti pada gambar 3.6.



Gambar 3.6 Algoritme HMAC

Jika tanda tangan tidak cocok maka data kemungkinan diubah, sebaliknya jika cocok maka data tidak berubah atau valid. Dengan kata lain, HMAC hanya berguna jika dipakai pada 1 layanan yang membuat dan menggunakan tokennya sendiri. Karena siapa saja yang memiliki *secretKey* dapat membuat token palsu.

Pada JWT juga terdapat algoritme asimetris seperti RSA256 yang menggunakan *privateKey* untuk menandatangani data. Sedangkan untuk memverifikasi sebuah data, RSA256 menggunakan publicKey seperti pada gambar 3.7.



Gambar 3.7 Algoritma RSA256

Jika tanda tangan tidak cocok maka data kemungkinan diubah, sebaliknya jika cocok maka data tidak berubah atau valid. Dengan kata lain RSA256 dapat

digunakan di layanan publik atau multiplatform karena untuk membuat token palsu dibutuhkan *privateKey* yang sulit didapatkan.

Dengan memanfaatkan dua algoritme tersebut, maka dapat dilakukan pemalsuan token dengan cara memanipulasi mengganti algoritme RSA256 pada sebuah token menjadi HMAC. Server yang seharusnya menerima token dengan algoritme RSA256 akan menerima token dengan algoritme HMAC. Sehingga siapa saja yang memiliki *publicKey* akan dapat menghasilkan token palsu. Misalkan JWT token asli yang dihasilkan dengan menggunakan *signature* algoritma RSA256 dapat didekode menggunakan base64Url sehingga didapatkan *header* dan *payload*. Kemudian ubah algoritma pada *header* menjadi HS256 kemudian enkode kembali.

Sama halnya dengan *payload*, isinya dapat diubah sesuai keinginan kemudian didekode kembali. Setelah itu data ditandatangani menggunakan algoritma HMAC dengan varian SHA256 menggunakan *publicKey*. *PublicKey* yang digunakan adalah *publicKey* yang sebelumnya digunakan untuk memverifikasi pada algoritma RSA256 dalam bentuk *hex* ASCII. Kemudian token palsu dengan algoritme HS256 yang dihasilkan dapat dicek validitasnya seperti pada gambar 3.8

```
Valid JWT: stdClass Object
(
    [iss] => http://demo.sjoerdlangkemper.nl/
    [iat] => 1571796168
    [exp] => 1571796288
    [data] => stdClass Object
        (
            [hello] => world
        )

)
```

Gambar 3.8 Validitas Token Palsu

Pemalsuan token juga dapat dilakukan dengan memanfaatkan *secretKey* yang digunakan oleh HMAC untuk menandatangani dan memverifikasi token. Jika *secretKey* tidak terlalu kuat, maka keamanannya mungkin dipatahkan menggunakan *brute-force attack* atau *dictionary attack*. Dengan mencoba beberapa kemungkinan *secretKey* pada JWT dan mengecek validitas sebuah token, maka dapat ditemukan *secretKey* yang digunakan. Hal ini dapat dilakukan tanpa harus terhubung pada server, setelah JWT diperoleh. Terdapat beberapa *tool* yang dapat digunakan untuk mematahkan HMAC seperti JwtCrack, Jwtbrute, dan JohnTheRipper.

3.2 Studi Literatur

Pada tahap penelitian ini dilakukan kajian mengenai API, rancangan API yang baik, arsitektur REST sebagai gaya arsitektur dan pendekatan komunikasi, implementasinya pada RESTful API, Platform-Agnostic SEcurity TOken (PASETO) sebagai mekanisme autentikasi user berbasis token menggunakan versi PASETO *v2*.

3.3 Analisis Kebutuhan

Pada tahap penelitian ini dilakukan analisis kebutuhan yang dibutuhkan selama penelitian meliputi:

3.3.1 Analisis Kebutuhan Perangkat Keras

Perangkat keras yang digunakan dalam penelitian ini adalah satu buah laptop sebagai platform untuk implementasi server RESTful API. Selain itu, laptop juga sebagai pihak yang berperan untuk menghasilkan token dan mengecek validitas sebuah token. Spesifikasi laptop yang digunakan adalah sebagai berikut:

- Model : ASUS X450JN
- *Operating System* : Windows 10 Pro Version 1903 Build 18362.418
- *Processor* : Intel(R) Core(TM) i7-4710 HQ CPU@ 2.50GHz
- RAM : 4 GB
- Jumlah : 1

3.3.2 Analisis Kebutuhan Perangkat Lunak

Perangkat lunak yang akan digunakan dalam penelitian ini, yaitu:

1. NodeJS

   Berfungsi sebagai *compiler* kode program yang digunakan untuk menjalankan API
2. MongoDB Atlas

   Berfungsi sebagai platform *database* yang digunakan untuk menampung skema *user*
3. Babel

   Berfungsi sebagai *compiler* kode program yang digunakan untuk menjalankan Javascript pada *node*
4. Express

   Berfungsi untuk membantu mengubah kode program Ecmascript6 (ES6) menjadi ES5, dikarenakan terdapat beberapa fitur ES6 yang tidak dimengerti oleh *node* dan browser. Sehingga Express dibutuhkan agar Babel menjalankan kode program yang dapat dimengerti oleh *node* dan browser.
5. Postman

   Berfungsi sebagai alat untuk mengakses API *route*
6. JWT

   Berfungsi sebagai pembanding mekanisme keamanan yang akan diimplementasi pada RESTful API.

7. PASETO

   Berfungsi sebagai mekanisme keamanan yang akan diimplementasi pada RESTful API.

8. Python 3.7.2

   Berfungsi sebagai *compiler* kode program yang digunakan untuk menjalankan PASETO

9. Libsodium

   Berfungsi sebagai *library* yang digunakan untuk algoritma pada PASETO.

10. Pysodium

    Berfungsi sebagai *wrapper* Libsodium agar dapat digunakan pada bahasa pemrograman Python 3.7.2.

3.3.3 Identifikasi Aktor

Aktor merupakan entitas yang berasal dari luar sistem atau entitas eksternal yang berinteraksi dengan sistem. Dalam sistem ini, terdapat dua aktor yang dijelaskan pada tabel 4.1.

Tabel 3.1 Aktor

No.	Aktor	Deskripsi
1.	*User*	Merupakan klien dengan *role user* yang hanya dapat mengakses informasi tentang klien itu sendiri
2.	Admin	Merupakan klien dengan *role* admin yang dapat mengakses informasi seluruh klien yang ada

3.3.4 Kebutuhan Fungsional

Kebutuhan fungsional sistem adalah seluruh fitur atau komponen yang diharapkan oleh *user* yang wajib dijalankan oleh sebuah sistem dalam situasi tertentu. Kebutuhan fungsional dari implementasi autentikasi berbasis token menggunakan PASETO sebagai mekanisme autentikansi RESTful API adalah sebagai berikut:

1. Klien dapat melakukan registrasi ke dalam sistem dengan memasukkan data nama, *email*, *password*, dan *role*.
2. Klien dapat melakukan *login* ke dalam sistem dengan memasukkan data email dan *password*
3. Server RESTful API dapat menghasilkan token dalam bentuk cookie yang dikirimkan kepada klien jika autentikasi (registrasi/*login*) berhasil
4. Token yang dihasilkan oleh server memiliki waktu kedaluwarsa sebanyak 5 menit

5. Klien dapat mengakses route /me berisi informasi tentang klien itu sendiri dengan mengirimkan token dalam bentuk *cookie* sebagai bukti autentikasi
6. Server RESTful API dapat memvalidasi token yang dikirimkan oleh klien berupa *cookie*
7. Klien dengan *role* admin dapat mengakses route /allData berisi informasi seluruh klien yang terdaftar ke dalam sistem
8. Klien dapat melakukan *logout* dengan menghilangkan *cookie*

3.3.5 Kebutuhan Nonfungsional

Kebutuhan nonfungsional sistem adalah kebutuhan sistem yang menitikberatkan pada properti perilaku yang dimiliki oleh sistem seperti kualitas atau batasan dari sistem itu sendiri. Kebutuhan nonfungsional secara umum dapat dikategorikan menjadi 4 jenis, yaitu *usability*, *portability*, *reliability*, *supportability*. Kebutuhan nonfungsional dari implementasi autentikasi berbasis token menggunakan PASETO sebagai mekanisme autentikansi RESTful API adalah *reliability*. Sistem menyediakan *hashing password* menggunakan bcrypt.

3.4 Perancangan Sistem

Pada tahap ini, sistem akan dirancang berdasarkan gambaran umum sistem yang didasari oleh kebutuhan sistem baik fungsional dan nonfungsional. Perancangan ini merupakan langkah-langkah operasi dalam pengolahan data dan prosedur untuk mendukung operasi sistem. Perancangan arsitektur sistem akan dimodelkan dalam bentuk topologi. Sedangkan perancangan server RESTful API dan mekanisme PASETO akan dimodelkan dalam bentuk *Unified Modelling Language* (UML). Pemodelan UML akan berisi *use case diagram* yang memetakan interaksi yang dapat dilakukan oleh aktor sebagai entitas luar sistem dengan server RESTful API. Selain itu UML juga akan berisi *sequence diagram* untuk memetakan aliran data antar objek.

3.4.1 Perancangan Arsistektur Sistem

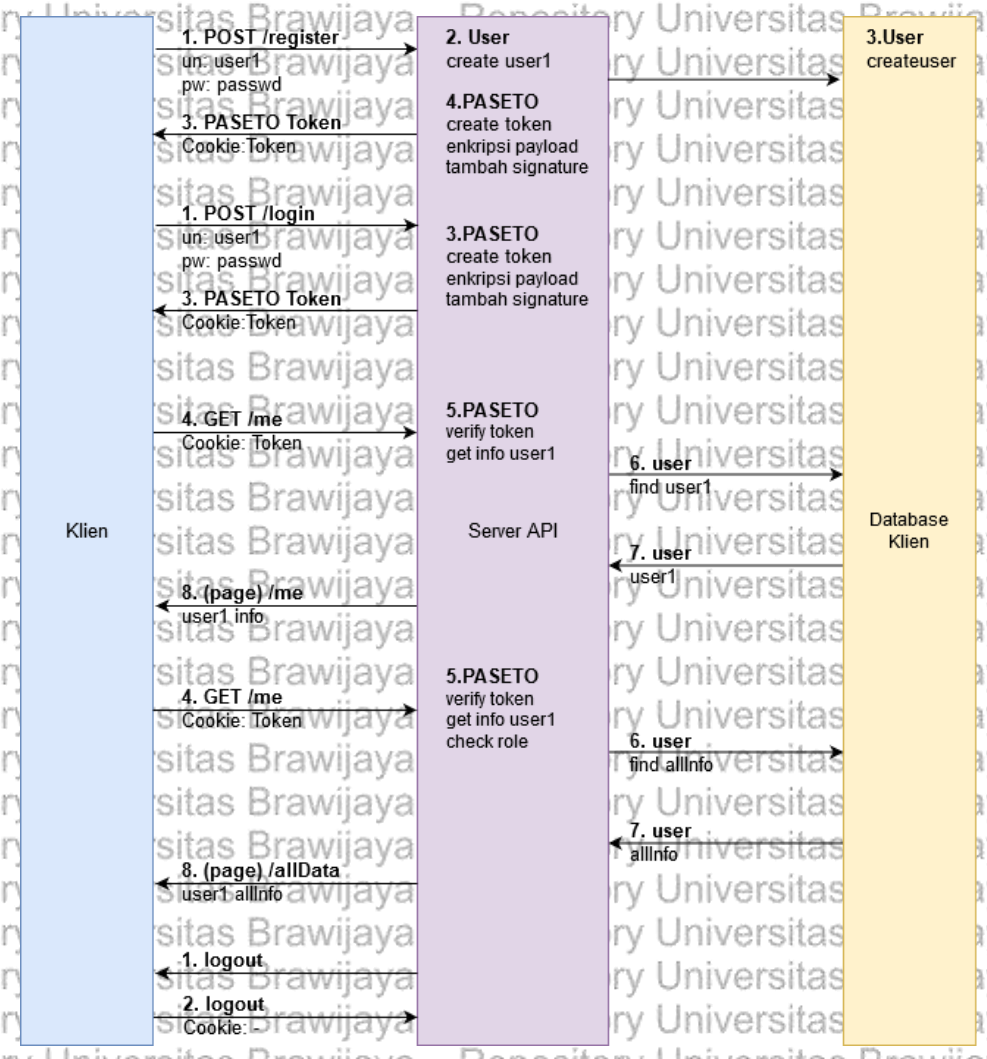
Sistem ini memiliki 3 entitas, yaitu server RESTful API, Postman, dan *database*. Server RESTful API merupakan server yang menangani setiap permintaan dari klien dengan mempertimbangkan role klien pada sistem. Server RESTful API memiliki skema *user* yang akan dihubungkan dengan *database* untuk menangani permintaan klien yang berhubungan dengan *database*. Server RESTful API menggunakan arsitektur RESTful API, sehingga server mampu berinteraksi dengan klien seperti autentikasi dan akses rute tertentu melalui *HTTP Request*.

Postman adalah *tool* yang digunakan oleh klien untuk mengakses rute yang disediakan oleh server RESTful API. Untuk pendaftaran klien baru, klien diharuskan untuk mengisi data klien berupa nama, *email*, *password*, dan *role*. Untuk *login*, klien diharuskan mengisi *email* dan *password*. Ketika autentikasi (*register*/*login*) dilakukan, maka klien akan menerima token dalam bentuk *cookie*. *Cookie* ini akan digunakan sebagai syarat untuk melakukan permintaan berupa pengaksesan

informasi klien itu sendiri dan permintaan informasi seluruh klien jika klien adalah admin.

Database yang digunakan adalah MongoDB Atlas yang akan dihubungkan dengan server RESTful API menggunakan Mongoose. *Database* digunakan untuk menampung seluruh informasi yang diberikan oleh server RESTful API. *Database* juga digunakan untuk memberikan data klien jika diminta.

Sistem ini menggunakan PASETO sebagai mekanisme keamanan menggantikan JWT. Alur sistem dapat dilihat pada gambar 3.9. Klien akan mendaftarkan *user* kepada server API dan server API mendaftarkan info tersebut ke dalam *database*. PASETO akan menghasilkan token menggunakan *username* dan *password*. Kemudian PASETO mengembalikan token yang memiliki batas waktu kepada klien, kemudian disimpan klien selama masih berlaku. Klien mencoba akses rute tertentu menggunakan token yang telah disimpan klien. Server API memverifikasi token, kemudian mencari klien dalam *database*. Jika ditemukan maka rute yang diminta diberikan pada klien.



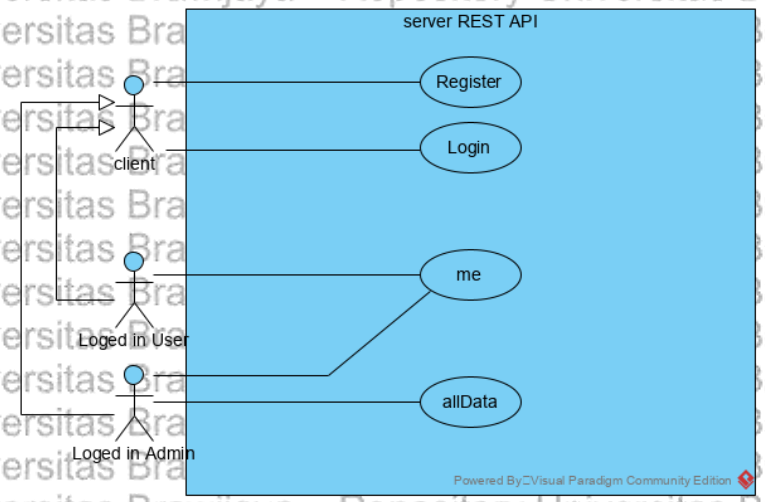
Gambar 3.9 Arsitektur Sistem

3.4.2 Perancangan Server RESTful API

Pada server RESTful API ini, harus mampu menangani *request* yang kirim client. Permintaan yang dimaksud adalah sebagai berikut:

1. POST /api/auth/register berupa pendaftaran klien baru yang diintegrasikan dengan PASETO untuk pembuatan token digunakan sebagai syarat pengaksesan rute tertentu.
2. POST /api/auth/login yaitu klien *login* ke dalam sistem untuk mendapatkan token yang digunakan sebagai syarat pengaksesan rute tertentu
3. GET /api/auth/me, berupa permintaan informasi klien itu sendiri dengan melampirkan token sebagai syarat akses rute.
4. GET /api/aut/allData permintaan informasi seluruh klien dengan melampirkan token sebagai syarat akses rute dan memiliki *role* admin
5. GET /logout berupa permintaan untuk menghapus token dari *cookie* sehingga perlu dilakukan proses *login* lagi agar dapat mengakses rute lain.

Pada Gambar 3.10 dijelaskan bahwa klien dapat mendaftarkan diri dan *login*. Klien yang mendaftarkan diri atau *login* dapat diidentifikasi statusnya sebagai klien biasa atau admin dari token yang akan diterima. Klien biasa hanya dapat mengakses rute me. Sedangkan klien yang statusnya diidentifikasi sebagai admin dapat mengakses rute allData.



Gambar 3.10 Use Case Diagram Server

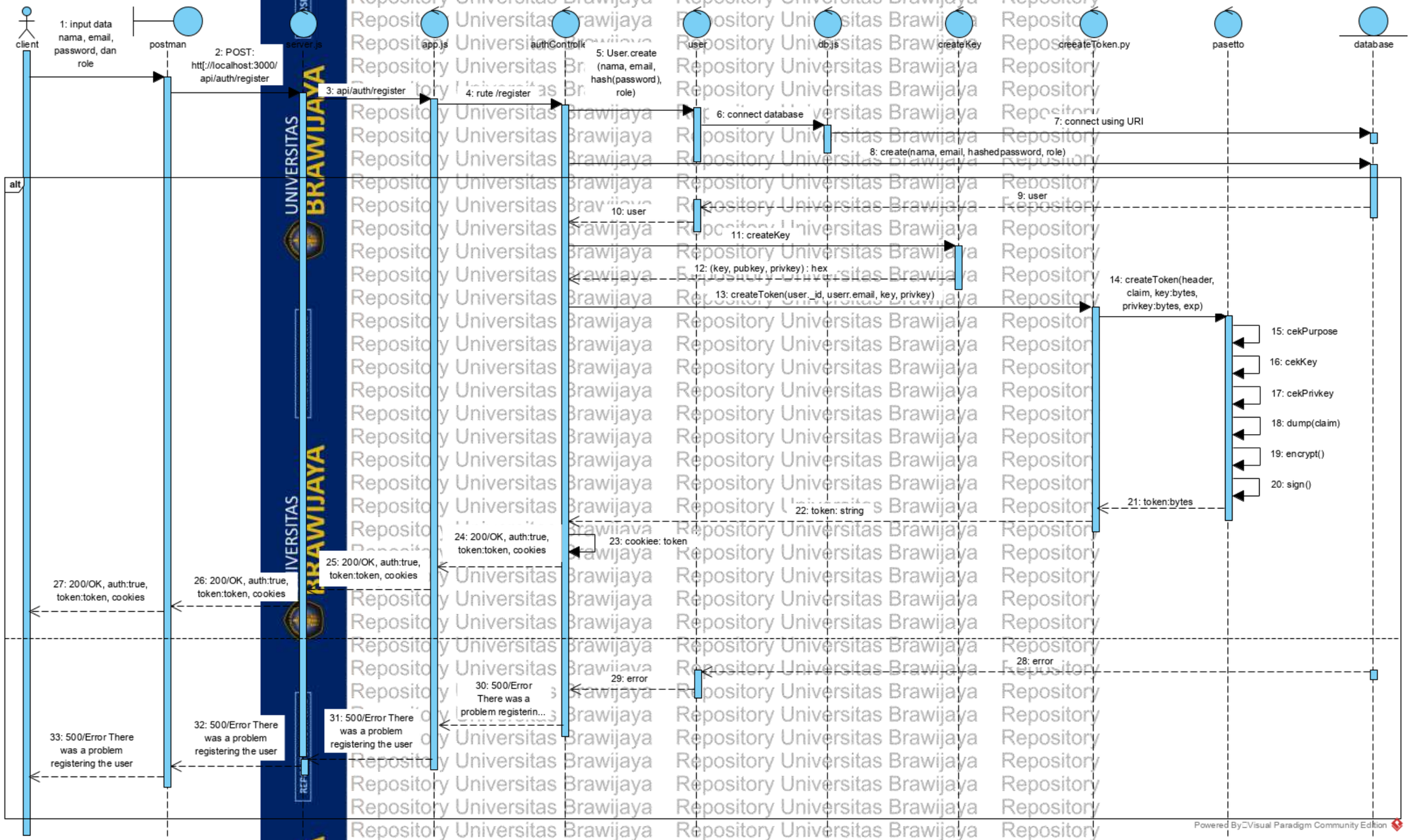
Proses aliran data yang terjadi ketika klien mendaftarkan diri dapat dilihat pada *sequence diagram* pada Gambar 3.11. Ketika klien ingin mendaftarkan diri, maka klien dapat memberi masukan berupa data nama, *email*, *password*, dan *role* ke dalam Postman sebagai *tool* yang digunakan untuk mengakses server RESTful API. Data tersebut ditujukan pada alamat http://localhost:3000/api/auth/register dan *method* POST. Permintaan tersebut akan diterima oleh server.js kemudian meneruskannya pada app.js dengan rute api/auth/register ke AuthController.js. AuthController.js akan memproses permintaan pada rute register dengan melakukan *hashing* pada *password* kemudian memasukkan seluruh data yang

diterima ke dalam model skema data pada user.js. Skema tersebut akan dikirim pada *database* dengan cara menghubungi db.js agar koneksi dengan *database* terjalin menggunakan URI. Setelah koneksi berhasil, maka skema data dikirimkan. Setelah itu *database* akan mengirimkan nilai kembalian berupa model data *user* atau *error* jika terjadi kesalahan. Jika terjadi kesalahan maka server akan mengembalikan status HTTP 500 dengan pesan "*There was a problem registering the user*". Jika berhasil maka server akan membuat kunci untuk pembentukan token. Kemudian kunci, *email* dan userId akan dikirimkan untuk pembuatan token menggunakan PASETO.

Untuk membuat sebuah token paseto dengan versi v2.custom maka dibutuhkan 2 kunci pkey untuk enkripsi dan privkey untuk menghasilkan signature yang telah dihasilkan sebelumnya. Permintaan pembuatan token dari createToken.py akan diproses oleh paseto.py dengan cara mengecek *purpose* yang dilampirkan oleh permintaan tersebut. Jika *purpose* yang dilampirkan tidak tersedia, maka proses dihentikan. Setelah itu, paseto.py akan mengecek apakah 2 kunci yang dibutuhkan dilampirkan atau tidak. Jika kunci yang dilampirkan tidak ada, maka proses dihentikan. Jika semua syarat terpenuhi, maka paseto akan menambahkan masa berlaku sebuah token ke dalam *claim* yang berisi data-data yang akan digunakan untuk pembuatan token seperti user._id, *email,* dan *role* sesuai yang dilampirkan. Data-data ini akan dimasukkan dalam format json yang dienkode dengan format utf-8.

Setelah itu maka dilakukan proses enkripsi untuk menghasilkan token dengan melampirkan *claim*, pkey, dan footer jika ada. Untuk melakukan proses enkripsi dibutuhkan *nonce* sebesar 24 *bytes*. *Nonce* dihasilkan dengan melakukan proses *hashing* menggunakan algoritme Blake2b pada *claim* sebagai *plaintext*. Proses ini menggunakan key yang berupa angka acak sebesar 24 *bytes*. Setelah itu dilakukan proses enkripsi untuk menghasilkan *ciphertext* menggunakan algoritme XChaCha20-Poly1305 pada *claim* sebagai *plaintext*. Selain itu dibutuhkan juga *header* v2.local, *nonce,* dan *footer* yang dikemas menggunakan *Pre-Authentication Encoding* dan *nonce* sebagai angka unik. Proses ini menggunakan key berupa pkey yang telah dilampirkan sebelumnya. Token kemudian dihasilkan dengan menggabungkan *header,* hasil enkode gabungan dari nonce dan *ciphertext.* Kemudian ditambahkan *footer* jika ada. Ketiganya dipisahkan tanda titik sebagai *separator*.

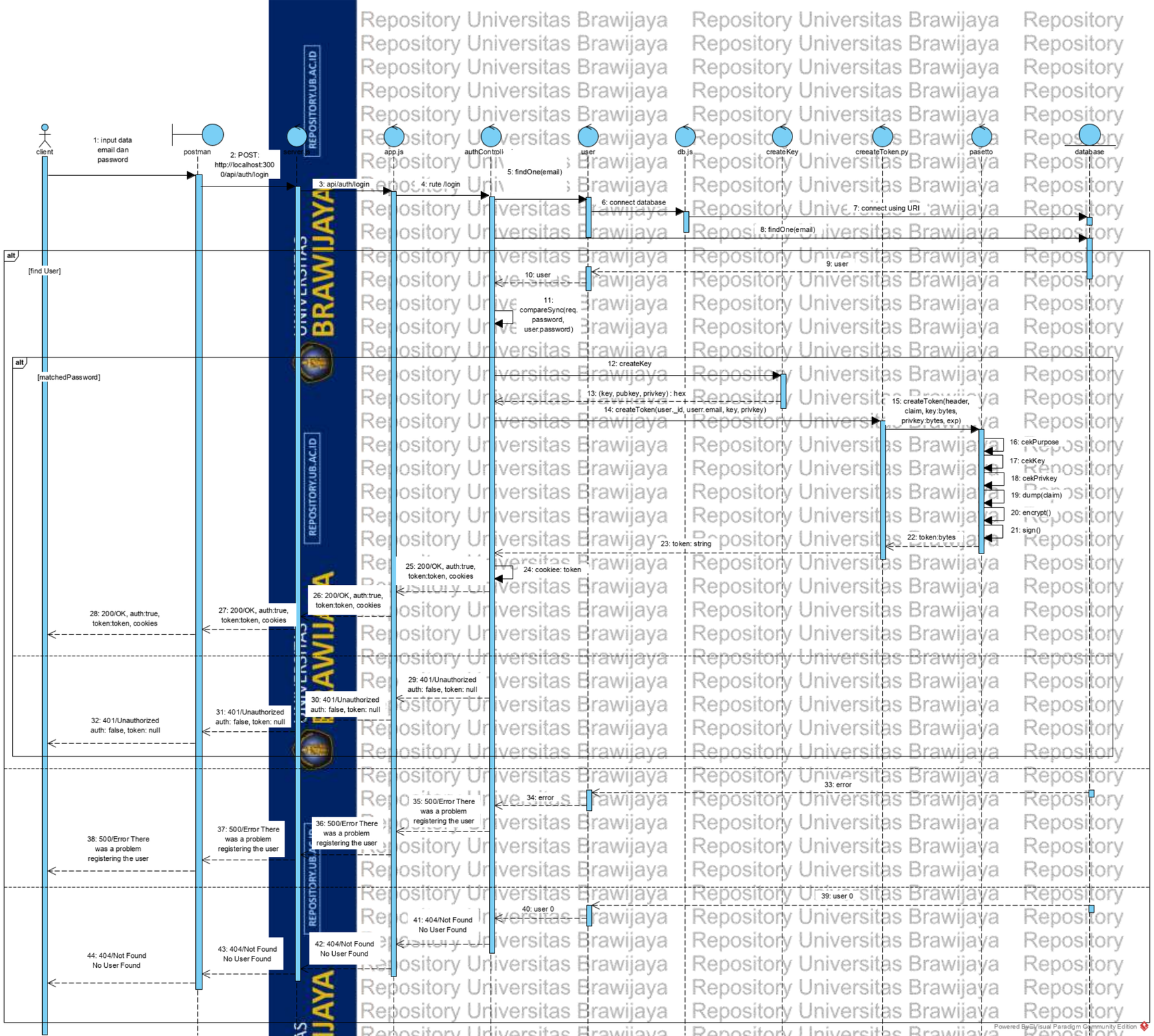
Setelah itu token yang dihasilkan akan ditambahkan *signature* sebesar 64 *bytes* menggunakan algoritme Ed25519 dengan sk sebagai *secret key. Plaintext* yang ditandatangani adalah header v2.public, token yang dihasilkan pada proses sebelumnya, dan *footer* jika ada yang diproses menggunakan *Pre-Authentication Encoding.* Token kemudian dihasilkan menggabungkan *public header* dan hasil enkode dari gabungan token sebelumnya dan signature yang sudah dihasilkan. Kemudian ditambahkan *footer* jika ada. Ketiganya dipisahkan tanda titik sebagai *separator*. Token akan dikirimkan kembali dalam bentuk *string* ke authController. Dan token akan dimasukkan ke dalam *cookie* bernama userData. Setelah itu server akan merespon dengan status HTTP 200 OK dengan pesan "auth: True" dan token.



Gambar 3.11 Sequence Diagram Register

Repository Universitas Brawijaya

Proses aliran data yang terjadi ketika klien *login* dapat dilihat pada *sequence diagram* pada Gambar 3.12. Ketika klien ingin *login*, maka klien dapat memberi masukan berupa data *email* dan *password* ke dalam Postman sebagai *tool* yang digunakan untuk mengakses server RESTful API. Data tersebut ditujukan pada alamat http://localhost:3000/api/auth/login dan *method* POST. Permintaan tersebut akan diterima oleh server.js kemudian meneruskannya pada app.js dengan rute api/auth/login ke AuthController.js. AuthController.js akan memproses permintaan pada rute *login* dengan melakukan *hashing* pada *password* kemudian mencari data pada *database* berdasarkan email menggunakan model skema data pada user.js. Skema tersebut akan dikirim pada *database* dengan cara menghubungi db,js agar koneksi dengan *database* terjalin menggunakan URI. Setelah koneksi berhasil, maka pencarian dapat dilakukan. Setelah itu *database* akan mengirimkan nilai kembalian berupa model data *user* atau *error* jika terjadi kesalahan. Jika terjadi kesalahan maka server akan mengembalikan status HTTP 500 dengan pesan "*There was a problem registering the user*". Jika berhasil maka *user* akan dicek isinya. Jika kosong maka server akan merespon dengan status HTTP 404 dengan pesan "*No User Found*". Jika *user* tidak kosong maka bandingkan *password* yang dimasukkan oleh *user* dengan *password* dari *database* dalam bentuk *hashing*. Jika tidak sama, server akan merespon dengan status 401 dengan pesan "*auth:false token:null*". Jika sama maka server akan membuat kunci untuk pembentukan token. Kemudian kunci, *email* dan *userId* akan dikirimkan untuk pembuatan token menggunakan PASETO. Proses pembuatan token sama dengan yang terjadi dengan sebelumnya. Kemudian token akan dimasukkan ke dalam *cookie* bernama userData. Setelah itu server akan merespon dengan status HTTP 200 OK dengan pesan "*auth: True*" dan token.



Gambar 3.12 Sequence Diagram Login

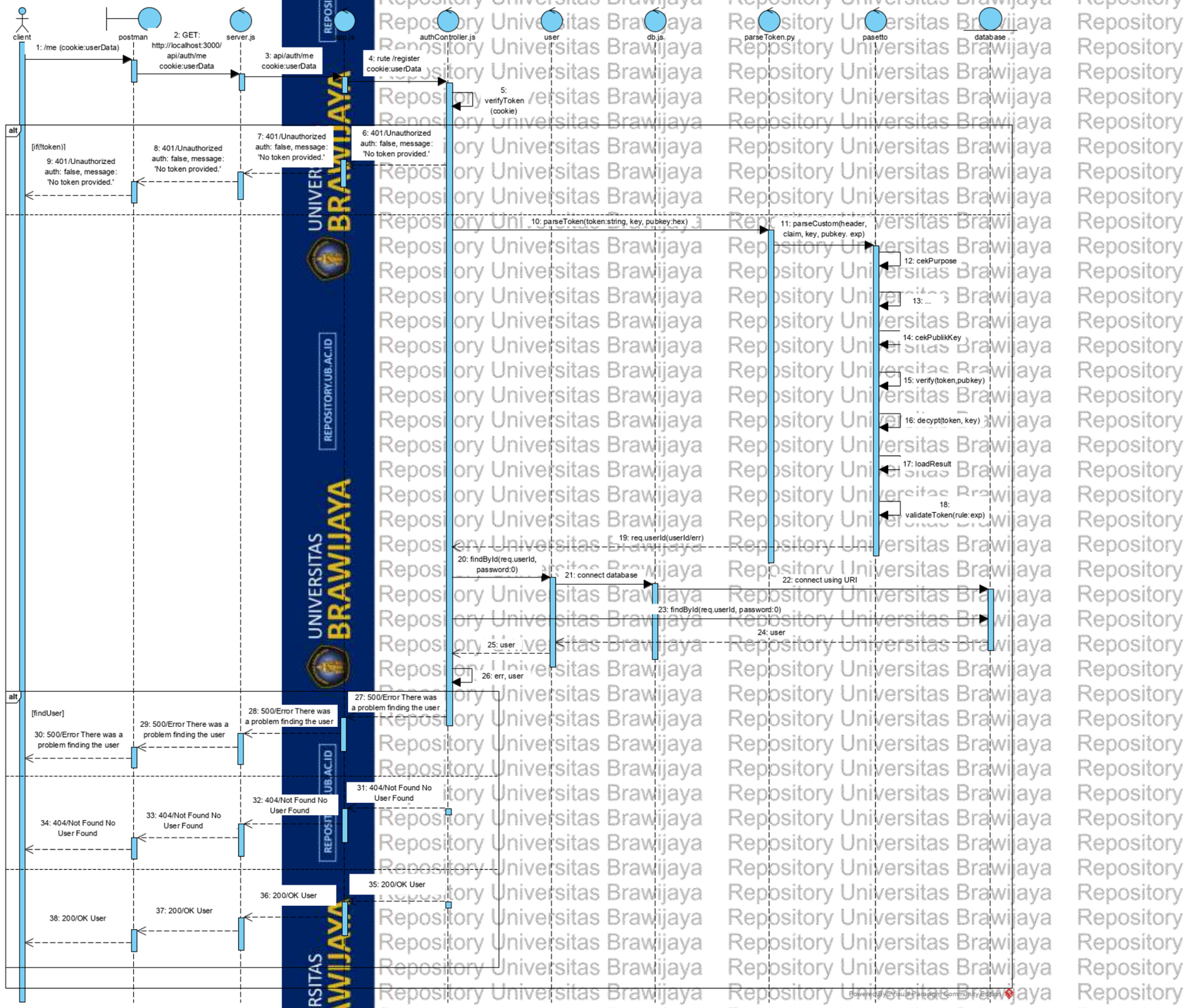
Proses aliran data yang terjadi ketika klien ingin mengakses data diri dapat dilihat pada *sequence diagram* pada Gambar 3.13. Ketika klien mengakses data diri, maka klien dapat menggunakan alamat http://localhost:3000/api/auth/me, *cookies* yang diterima sebelumnya, dan *method* GET pada Postman. Permintaan tersebut akan diterima oleh server.js kemudian meneruskannya pada app.js dengan rute api/auth/me ke AuthController.js. AuthController.js akan memproses permintaan pada rute me dengan melakukan verifikasi pada token yang terdapat pada *cookie*. Jika token kosong maka server akan merespon dengan status 401 dan pesan "*auth:false, message: No Token Provided*". Jika token tidak kosong maka token dikirimkan ke parseToken.py beserta dengan kunci untuk dicek. ParseToken akan mengirimkan ke pasetoToken untuk dicek keaslian token menggunakan paseto.

Untuk menguraikan sebuah token, maka dibutuhkan proses pengecekan *purpose* sebuah token. Pengecekan dilakukan dengan membandingkan *purpose* yang tertera pada token dan *purpose* yang diharapkan oleh sistem. Jika tidak cocok maka proses dihentikan. Hal ini berbeda dengan yang terjadi pada JWT, dimana JWT selalu percaya dengan header yang diberikan pada token. Kemudian dilakukan pengecekan kunci yang dilampirkan apakah ada key untuk enkripsi dan pkey untuk verifikasi. Jika tidak ada, maka proses dihentikan. Setelah semua syarat terpenuhi maka token diverifikasi kemudian didekripsi.

Verifikasi token dimulai dengan mengambil *header* pada token sepanjang public_header yang telah didefinisikan sebelumnya, dimana 2 indeks pertama adalah versi *header*. Kemudian dibandingkan kembali apakah header adalah v2.public. Jika tidak, maka proses akan dihentikan. Setelah itu token dipisahkan menggunakan tanda titik sebagai *separator*. Setelah dipisahkan maka bagian-bagian tersebut dicek, jika berjumlah empat maka bagian terakhir adalah footer. Bagian kedua merupakan gabungan token yang dienkripsi sebelumnya dan *signature*. Bagian kedua ini akan dilakukan proses enkode agar dapat dibaca. Signature dipisahkan dengan mengambil 64 *bytes* terakhir sedangkan token enkripsi sisanya. Token akan dicek keasliannya dengan melampirkan *signature, plaintext berupa header* token, token yang masih terenkripsi, dan footer yang diproses menggunakan *Pre-Authentication Encoding,* dan pkey sebagai *key.* Jika *signature* tidak valid, maka proses dihentikan.

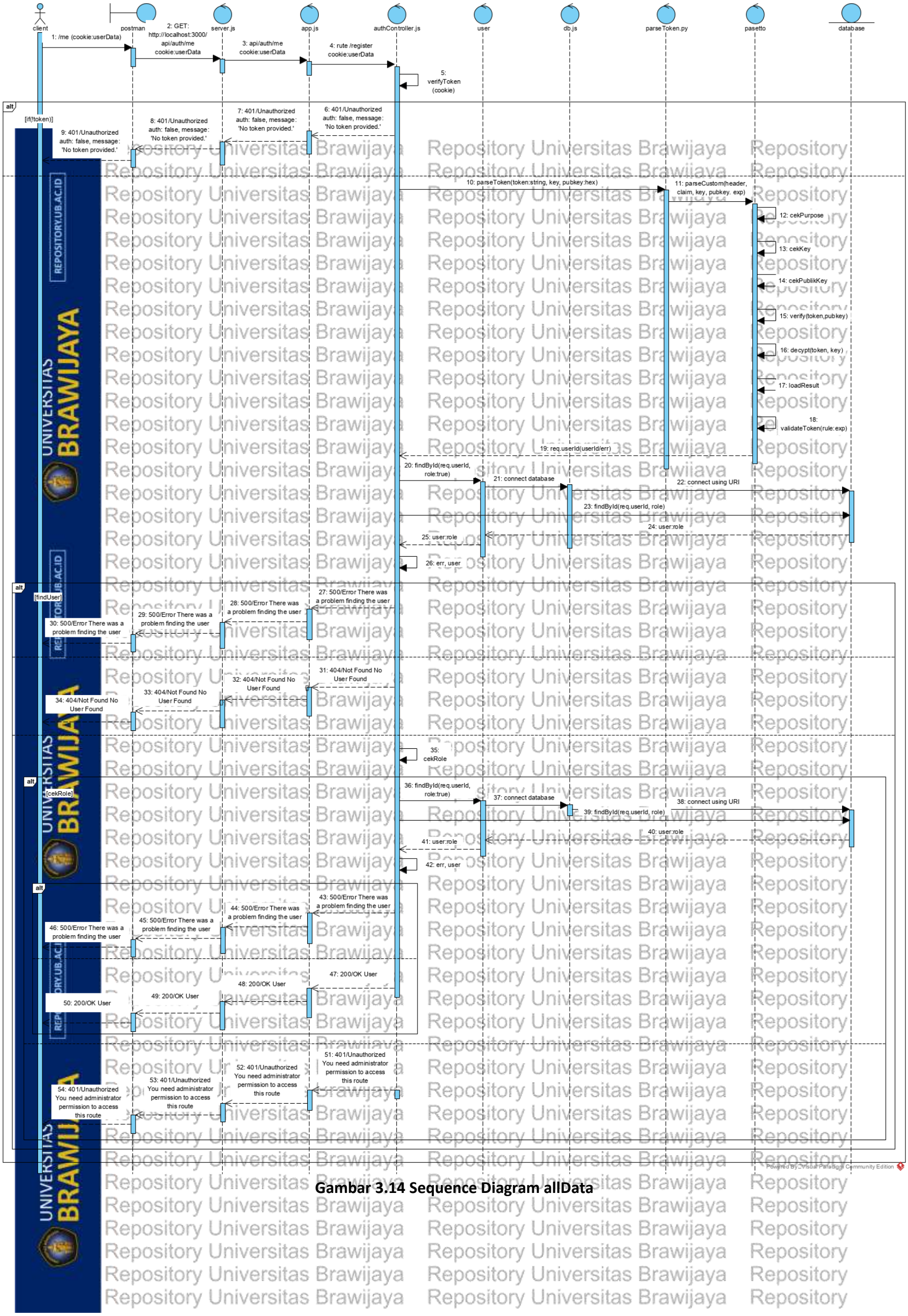
Selanjutkan dilakukan proses dekripsi pada token. Dimulai dengan memisahkan token ke dalam beberapa bagian menggunakan tanda tititk sebagai *separator*. Jika berjumlah empat maka footer terdapat pada bagian terakhir. Selanjutnya header token diambil dari token sepanjang local_header, dimana 2 indeks pertama adalah versi *header.* Bagian kedua kemudian dienkode agar dapat dibaca. Bagian kedua merupakan gabungan *ciphertext* dan *nonce*. *Nonce* diambil dari 24 *bytes* akhir dan ciphertext sisanya. *Ciphertext* kemudian didekripsi menggunakan algoritme XChaCha20-Poly1305 dengan melampirkan ciphertext, hasil *Pre-Authentication Encoding* dari *header,* nonce*, dan footer.* Kemudian dilampirkan juga nonce yang didapatkan dan key yang dilampirkan. Setelah itu *plaintext* akan didapatkan.

Paseto akan memberikan nilai kembalian berupa userId berisi *id user* pada *database* atau nilai *error*. Kemudian AuthController akan mencari *user* pada database berdasarkan userId. Jika error maka server akan merespon dengan nilai status 500 dan pesan "*There was a problem finding the user*". JIka tidak *error* namun *user* kosong maka server akan merespon dengan status HTTP 404 dengan pesan *"No user found"*. Jika user tidak kosong maka server akan merespon dengan status 200 dan pesan data klien itu sendiri.



Gambar 3.13 Sequnce Diagram Rute Me

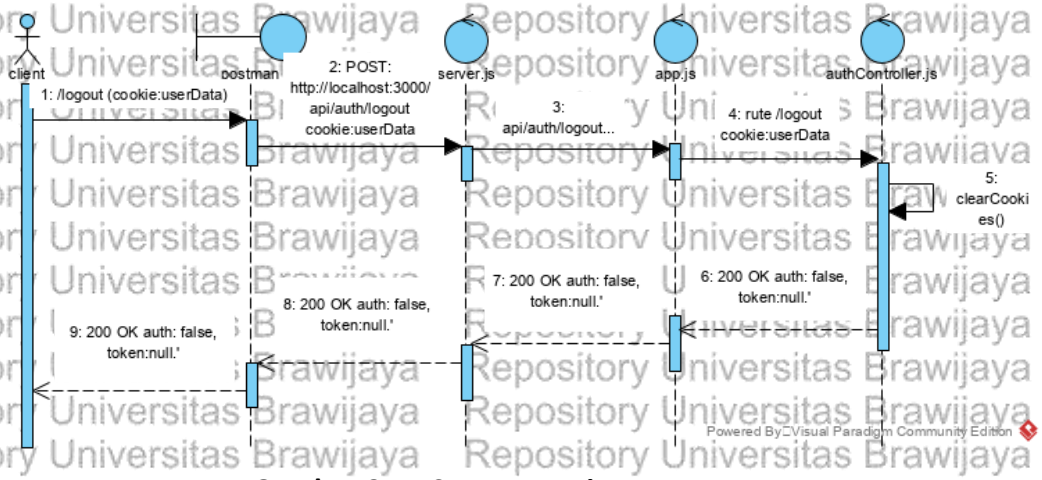
Proses aliran data yang terjadi ketika klien ingin mengakses data seluruh klien dapat dilihat pada *sequence diagram* pada Gambar 3.14. Ketika klien mengakses data seluruh klien, maka klien dapat menggunakan alamat http://localhost:3000/api/auth/allData, *cookies* yang diterima sebelumnya, dan *method* GET pada Postman. Permintaan tersebut akan diterima oleh server.js kemudian meneruskannya pada app.js dengan rute api/auth/me ke AuthController.js. AuthController.js akan memproses permintaan pada rute me dengan melakukan verifikasi pada token yang terdapat pada *cookie*. Jika token kosong maka server akan merespon dengan status 401 dan pesan "*auth:false, message: No Token Provided*". Jika token tidak kosong maka token dikirimkan ke parseToken.py beserta dengan kunci untuk dicek. parseToken akan mengirimkan ke pasetoToken untuk dicek keaslian token menggunakan paseto. Proses verifikasi token sama dengan yang terjadi pada saat mengakses informasi diri pada klien. Paseto akan memberikan nilai kembalian berupa userId berisi id *user* pada *database* atau nilai *error*. Kemudian AuthController akan mencari *user* pada *database* berdasarkan userId dengan mengembalikan *role* klien saja. Jika *error* maka server akan merespon dengan nilai status 500 dan pesan "*There was a problem finding the user*". JIka tidak *error* namun *user* kosong maka server akan merespon dengan status HTTP 404 dengan pesan "*No user found*". Jika *user* tidak kosong maka server akan mengecek *role*. Jika *role* bukan admin maka server akan merespon dengan status 401 dengan pesan "*You need administrator permission to access this route*". Jika *role* adalah admin maka server akan merespon dengan status 200 dan pesan data seluruh klien.



Gambar 3.14 Sequence Diagram allData

Repository Universitas Brawijaya

Proses aliran data yang terjadi ketika klien ingin keluar dari server *sequence diagram* pada Gambar 3.15. Ketika klien ingin keluar dari server, maka klien dapat menggunakan alamat http://localhost:3000/api/auth/logout, *cookies* yang diterima sebelumnya, dan *method* POSTpada Postman. Permintaan tersebut akan diterima oleh server.js kemudian meneruskannya pada app.js dengan rute api/auth/logout ke AuthController.js. AuthController.js akan memproses permintaan pada rute logout dengan menghapus isi *cookies* dan server akan merespon dengan status H. Jika token kosong maka server akan merespon dengan HTTP 200 dengan pesan "*auth:false, token: null*".



Gambar 3.15 Sequence Diagram Logout

3.4.3 Perancangan Pengujian

Pada penelitian ini, terdapat 3 pengujian yaitu pengujian fungsional, keamanan, dan kinerja.

1. Pengujian Fungsional dilakukan untuk mengetahui apakah:
   - PASETO dapat menghasilkan *message* berisi *version, purpose, payload*
   - Klien dapat register
   - Klien dapat *login* menggunakan token
   - Klien dapat mengakses rute menggunakan token pada *cookie*
   - Klien dapat mengakses rute menggunakan token pada *cookie* dengan *role* tertentu
   - Klien dapat *logout*
2. Pengujian Nonfungsional dilakukan untuk mengetahui apakah:
   - Sistem menyediakan *hashing password* menggunakan Bcrypt
3. Pengujian Keamanan dilakukan untuk mengetahui apakah:
   - Token memiliki waktu *expired* selama 5 menit
   - Klien tidak dapat terhubung apabila token invalid
   - Klien tertentu hanya dapat mengakses rute tertentu

- PASETO diserang menggunakan algoritme None
- PASETO diserang menggunakan eksploitasi algoritme simetris-asimetris

4. Pengujian Kinerja dilakukan untuk mengetahui:
   - Catatan waktu yang dibutuhkan klien untuk meminta token
   - Catatan waktu yang dibutuhkan untuk validasi token

3.5 Implementasi

Pada tahap penelitian ini, terdapat 3 tahapan implementasi yaitu sebagai berikut:

1. Implementasi struktur *file* yang akan digunakan untuk membuat *mock user* pada sebuah RESTful API untuk digunakan sebagai tempat *login* dan menerima permintaan *user*.
2. Implementasi PASETO yang dapat menghasilkan dan memverifikasi token dengan masa *expired* tertentu berdasarkan *username* dan *password*.
3. Menghubungkan struktur *file* RESTful API dengan PASETO untuk penerimaan token berdasarkan *username* dan *password*.

3.6 Pengujian dan Analisis

Pada tahap penelitian ini akan dilakukan pengujian yang telah dijelaskan pada perancangan pengujian untuk menunjukkan apakah sistem dapat berjalan sesuai dengan kebutuhan fungsional, keamanan dan tujuan yang telah diuraikan sebelumnya.

3.7 Pengambilan Kesimpulan

Setelah semua tahapan perancangan implementasi, pengujian, dan analisis yang diterapkan sudah selesai dilakukan maka akan ditarik kesimpulan. Kesimpulan diambil dari hasil pengujian dan analisis. Selanjutnya tahap akhir dari penulisan ini adalah saran yang akan menjadi perbaikan dan pertimbangan pada pengembangan selanjutnya.

BAB 4 IMPLEMENTASI

4.1 Gambaran Umum Sistem

Implementasi autentikasi berbasis token menggunakan PASETO sebagai mekanisme autentikansi RESTful API merupakan sebuah RESTful API yang membutuhkan token ketika klien ingin mengakses rute tertentu. Hal ini dibutuhkan agar klien tidak mengirim data *credential* setiap kali ingin mengakses rute tertentu. Mekanisme PASETO yang digunakan adalah PASETO versi dua dengan menggabungkan fungsi enkripsi dan *signature* dalam satu token.

Sistem ini memiliki 3 entitas saja, yaitu server RESTful API, Postman, dan *database*. Server RESTful API merupakan server yang menangani setiap permintaan dari client. Postman adalah tool yang digunakan oleh klien untuk mengakses rute yang disediakan oleh server RESTful API. Dan *database* yang akan dihubungkan dengan server RESTful API.

4.2 Implementasi

Pada tahap ini akan dijelaskan 10 komponen utama pada implementasi server RESTful API. Implementasi ini meliputi bagaimana server berjalan, terhubung dengan *database*, rute yang dimiliki, pembentukan, dan penguraian token. Kemudian dilanjutkan dengan implementasi PASETO sebagai mekanisme yang digunakan oleh server RESTful API untuk autentikasi.

4.2.1 Konfigurasi Aplikasi

Implementasi konfigurasi aplikasi diletakkan pada app.js menggunakan Node.js. App.js ini berguna untuk mengonfigurasi seluruh komponen dan menghubungkannya dengan server. Komponen tersebut akan didaftarkan pada app.js agar dapat digunakan oleh komponen lain jika diperlukan. Kode program pp.js dituliskan sebagai berikut:

Source Code 4.1: Konfigurasi App

```
1  var express = require('express');
2  var app = express();
3  var db = require('./db');
4  var cookieParser = require('cookie-parser');
5  app.use(cookieParser());
6  var AuthController = require('./auth/AuthController');
7  app.use('/api/auth', AuthController);
8  module.exports = app;
```

Penjelasan *source code* 1 adalah sebagai berikut:

Baris 1-2 : Menggunakan Express sebagai *framework* api

Baris 3 : Menggunakan variabel db untuk memanggil file db.js

Baris 4-5 : Menggunakan variabel *cookie-parser* untuk menggunakan cookie

Baris 6 : Menggunakan variabel AuthController untuk memanggil file auth/AuthController pada direktori ./auth

Baris 7 : Menggunakan /api/auth untuk mengakses AuthController

Baris 8 : Objek app menjadi *visible* jika ada program lain yang ingin membutuhkannya menggunakan require()

4.2.2 Database

Dengan berjalannya server maka kita membutuhkan *database* untuk dihubungkan menggunakan Mongoose. Mongoose adalah ORM (Object Relational Mapper) *tool* yang digunakan untuk menyederhanakan pengiriman data antara server API dan *database*. Mongoose memetakan data yang kita punya di server API dengan seperangkat aturan ketat yang ditetapkan *database*. Implementasi konfigurasi *database* diletakkan pada db.js. Kode program db.js dituliskan sebagai berikut:

Source Code 4.2: Konfigurasi Database

```
1 var mongoose = require('mongoose');
2 mongoose.connect('mongodb+srv://nicositorus:enel127026@databasetesti
3 ng-ujz8f.mongodb.net/test?retryWrites=true&w=majority', {
4 useNewUrlParser: true });
```

Penjelasan *source code* 2 adalah sebagai berikut:

Baris 1 : Menggunakan db.js untuk menyetor data ke dalam *database* menggunakan *library* Mongoose

Baris 2-4 : Menghubungkan server api dengan *database* mongoDB Atlas menggunakan *username* dan *password* pada databaseURI

4.2.3 Server

Implementasi server diletakkan pada server.js menggunakan Node.js yang digunakan untuk menjalankan server dengan memanggil app.js ke dalam sebuah variabel dan mendengarkan setiap permintaan yang ditujukan pada server API melalui *port* tertentu.

Source Code 4.3: Konfigurasi Server

```
1 var app = require('./app');
2 var port = process.env.PORT || 3000;
3 var server = app.listen(port,
4     function() {
5         console.log('Express server listening on port ' + port);
6 }
7 );
```

Penjelasan source code 3 adalah sebagai berikut:

Baris 1 : Memanggil app.js menggunakan variabel app

Baris 2 : Memilih *port* yang akan digunakan oleh app untuk berjalan

Baris 3-7 : App mendengarkan permintaan pada port yang telah ditentukan kemudian mencetak log pada terminal

4.2.4 Skema *User*

Implementasi skema *user* diletakkan pada direktori *user* pada kode user.js menggunakan Node.js untuk menangani model *blueprint* user yang akan digunakan oleh server API. Skema *user* akan di-*binding* dengan nama User kemudian di ekspor menggunakan modul dengan model Mongoose. Nama *user* ini akan digunakan oleh kode program lain untuk mengakses data dalam *database*.

Source Code 4.4: Skema User

```
1  var mongoose = require('mongoose');
2  var UserSchema = new mongoose.Schema({
3                   name: String,
4                   email: String,
5                   password: String,
6                   role:String
7              }
8  );
9  mongoose.model('User', UserSchema);
10 module.exports = mongoose.model('User');
```

Penjelasan *source code* 4 adalah sebagai berikut:

Baris 1 : Menggunakan variabel Mongoose untuk menampung *library* Mongoose

Baris 2-8 : Membuat skema untuk data *user* yang akan dikirim ke *database*

Baris 9 : Menggunakan nama User untuk model UserSchema yang digunakan pada Mongoose

Baris 10 : Objek User menjadi *visible* jika ada program lain yang ingin membutuhkannya menggunakan require()

4.2.5 Autentikasi *Controller*

Implementasi autentikasi *controller* diletakkan pada direktori auth yang berisi kode program authController.js menggunakan Node.js yang digunakan untuk mengontrol *flow* data dari *database* dan menuju *database* beserta mekanisme keamanan yang digunakan. auhController akan mendefinisikan objek route yang sesuai dengan HTTP *method*. Setiap rute akan menentukan *callback function* tertentu ketika menerima permintaan ke rute dan *method* HTTP yang telah ditentukan. Dengan kata lain server API akan mendengarkan permintaan yang cocok dengan rute dan *method* tertentu, kemudian memanggil *callback function* tertentu.

1. *Register*

Source Code 4.5: Register

```
1  router.post('/register',
2    function(req, res) {
3      var hashedPassword = bcrypt.hashSync(req.body.password, 8);
4      User.create({
5        name : req.body.name,
6        email : req.body.email,
7        password : hashedPassword,
8        role : req.body.role
9      },
```

Source Code 4.5: Register Lanjutan

```
10      function (err, user) {
11        if (err)
12          return res.status(500).send("There was a problem registering
13  the user.")
14          var spawn = require("child_process").spawn;
15          var process = spawn('python',["./auth/createKey.py"]);
16         process.stdout.on('data',
17          function (data) {
18            keyStr = data.toString().replace(/(\r\n|\n|\r)/gm, "");
19            allKey = keyStr.split("pisah");
20            mkey = allKey[0];
21           pkey = allKey[1];
22            skey = allKey[2];
23            var spawn = require("child_process").spawn;
24            var process = spawn('python',["./auth/createToken.py",
25            user._id,
26            req.body.email,
27            user.role,
28            key=mkey,
29           privkey=skey]);
30            process.stdout.on('data',
31            function (data) {
32             token = data.toString().replace(/(\r\n|\n|\r)/gm, "");
33             res.cookie('userData', token);
34             res.status(200).send({ auth: true, token: token});
35           }
36         )
37       }
38     )
39   }
40  );
41 }
42 );
```

Penjelasan *source code* 5 adalah sebagai berikut:

Baris 1 : Rute register menggunakan POST dan parameter /register sebagai rute yang dihubungkan ke parameter kedua

Baris 2 : Parameter function sebagai fungsi yang dijalankan jika rute register dipanggil

Baris 3 : *Hashing password* dengan *hash round* 2^{rounds}

Baris 4-9 : Menerima data yang diberikan pada server dan memasukkan ke dalam *database* menggunakan skema user

Baris 10 : Setelah data dimasukkan maka terdapat *callback function* berupa *error* atau sukses

Baris 11-12 : Jika *error* server merespon status 500 dengan pesan *"There was a problem registering the user."*

Baris 14-22 : Jika valid maka buat key dengan memanggil kode program createKey.py dan terima ke dalam bentuk *string*

menggunakan stdout kemudian pisahkan menjadi *key*, *public key*, dan *private key*.

Baris 23-32 : Buat token dengan mengirimkan user._id, email, user.role, dan kunci yang diperlukan pada kode program createToken.py kemudian terima token dalam bentuk *string* menggunakan stdout.

Baris 33 : Masukkan token ke dalam *cookie*

Baris 34 : Server merespon HTTPP 200/Ok autentikasi berhasil

2. *Login*

Source Code 4.6: Login

```
1  router.post('/login',
2  function(req, res) {
3    User.findOne({
4    email: req.body.email
5    },
6     function (err, user) {
7      if (err)
8       return res.status(500).send('Error on the server.');
9      if (!user)
10      return res.status(404).send('No user found.');
11     var  passwordIsValid  =  bcrypt.compareSync(req.body.password,
12 user.password);
13     if (!passwordIsValid)
14       return res.status(401).send({ auth: false, token: null });
15     var spawn = require("child_process").spawn;
16     var process = spawn('python',["./auth/createKey.py"]);
17     process.stdout.on('data',
18      function (data) {
19       keyStr = data.toString().replace(/(\r\n|\n|\r)/gm, "");
20       allKey = keyStr.split("pisah");
21       mkey = allKey[0];
22       pkey = allKey[1];
23       skey = allKey[2];
24       var spawn = require("child_process").spawn;
25       var process = spawn('python',["./auth/createToken.py",
26        user._id,
27        req.body.email,
28        user.role,
29        key=mkey,
30        privkey=skey]);
31        process.stdout.on('data',
32         function (data) {
33          token = data.toString().replace(/(\r\n|\n|\r)/gm, "");
34          res.cookie('userData', token);
35          res.status(200).send({ auth: true, token: token});
36         }
37        )
38      }
39     )
40   }
41 );
42 }
43 );
```

Penjelasan *source code* 6 adalah sebagai berikut:

Baris 1 : Rute *login* menggunakan POST dan parameter /login sebagai rute yang dihubungkan ke parameter kedua

Baris 2 : Parameter function sebagai fungsi yang dijalankan jika rute *login* dipanggil

Baris 3-5 : Cari *user* berdasarkan email

Baris 6 : Parameter function sebagai fungsi yang dijalankan setelah mencari user

Baris 7-8 : Jika *error* maka terdapat kesalahan pada server

Baris 9-10 : Jika sukses namun user tidak dapat ditemukan.

Baris 11-14 : Bandingkan *password* dan jika tidak sama maka *autentikasi* gagal

Baris 15-23 : Jika valid maka buat *key* dengan memanggil kode program createKey.py dan terima ke dalam bentuk *string* menggunakan stdout kemudian pisahkan menjadi key, *public key*, dan *private key*.

Baris 24-33 : Buat token dengan mengirimkan user._id, email, dan kunci yang diperlukan pada kode program createToken.py kemudian terima token dalam bentuk *string* menggunakan stdout.

Baris 34 : Masukkan token ke dalam *cookie*

Baris 35 : Server merespon HTTPP 200/Ok autentikasi berhasil

3. Informasi Klien

Source Code 4.7: Informasi Klien

```
1  router.get('/me',
2  (req, res, next) => {
3    verifyToken(req, res, next);
4  },
5  function(req, res, next) {
6    if(req.userId == "error"){
7      res.status(500).send("Token invalid");
8    }
9    else {
10     User.findById(req.userId,
11       { password: 0 },
12       function (err, user) {
13         if (err)
14           return res.status(500).send("There was a problem finding the
15 user.");
16         if (!user)
17           return res.status(404).send("No user found.");
18         res.status(200).send(user);
19       }
20     );
21   }
22 }
23 );
```

Penjelasan *source code* 7 adalah sebagai berikut:

Baris 1 : Rute me menggunakan GET dan parameter /me sebagai rute yang dihubungkan ke parameter kedua.

Baris 2-4 : Parameter function sebagai fungsi yang dijalankan untuk verifikasi token.

Baris 5 : Parameter function sebagai fungsi yang dijalankan setelah verifikasi token.

Baris 6 : Cek token valid atau tidak.

Baris 7-8 : Jika tidak valid, server merespon HTTPP 500 dengan pesan "*Token invalid*"

Baris 9-11 : Jika tidak, maka server informasi mengenai klien berdasarkan user_id.

Baris 12 : Parameter function sebagai fungsi yang dijalankan setelah mencari user

Baris 13-15 : Jika error, server merespon HTTPP 500 dengan pesan "*There was a problem finding the user.*"

Baris 16-17 : Jika sukses namun user tidak dapat ditemukan, server merespon dengan status 404 dengan pesan "*No user found.*"

Baris 14 : Jika user ada, server merespon HTTPP 200/Ok dan data klien

4. Informasi Seluruh Klien

Source Code 4.8: Informasi Seluruh Klien

```
1  router.get('/allData',
2  (req, res, next) => {
3      verifyToken(req, res, next);
4  },
5  function(req, res, next) {
6    if(req.userId == "error"){
7     res.status(500).send("Token Invalid");
8    }
9    else if(req.role == "admin"){
10     User.find({},
11       function (err, users) {
12         if (err)
13           return res.status(500).send("There was a problem finding the
14  users.");
15         res.status(200).send(users);
16       }
17     );
18    }
19    else{
20     res.status(401).send("You need administrator permission to access
21  this route");
22    }
23  }
24  );
```

Penjelasan *source code* 8 adalah sebagai berikut:

Baris 1 : Rute me menggunakan get dan parameter /me sebagai rute yang dihubungkan ke parameter kedua.

Baris 2-4 : Parameter function sebagai fungsi yang dijalankan untuk verifikasi token.

Baris 5 : Parameter function sebagai fungsi yang dijalankan setelah verifikasi token.

Baris 6 : Cek token valid atau tidak.

Baris 7-8 : Jika tidak valid, server merespon HTTPP 500 dengan pesan *"Token invalid"*

Baris 9 : Cek jika *role* pada token adalah admin

Baris 10 : Jika admin maka, cari informasi seluruh klien

Baris 11 : Parameter function sebagai fungsi yang dijalankan setelah mencari klien

Baris 12-14 : Jika *error*, server merespon HTTPP 500 dengan pesan "*There was a problem finding the user."*

Baris 15 : Server merespon HTTPP 200/Ok dan data klien

Baris 19-21 : Jika bukan admin maka server merespon status 401 *Unauthorized*

5. Verifikasi Token

Source Code 4.9: Verifikasi Token

```
1  var verifyToken = (req, res, next) => {
2   var token = req.cookies.userData;
3    if (!token)
4      return res.status(401).send({ auth: false, message: 'No token
5  provided.' });
6    var spawn = require("child_process").spawn;
7    var process = spawn('python',["./auth/parseToken.py", token, mkey,
8  pkey]);
9    process.stdout.on('data',
10    function (data) {
11      data = data.toString();
12      data = data.replace(/(\r\n|\n|\r)/gm, "");
13      parts = data.split(' ');
14      req.userId = parts[0];
15      req.email = parts[1];
16      req.role = parts[2];
17      next();
18    }
19   )
20 };
```

Penjelasan *source code* 9 adalah sebagai berikut:

Baris 1 : Fungsi *middleware* (aturan yang harus dilewati terlebih dahulu sebelum masuk ke dalam sebuah sistem atau keluar dari sebuah sistem) yang dipanggil untuk memverifikasi token.

Baris 2-5 : Cek token jika tidak ada

Baris 5 : Parameter function sebagai fungsi yang dijalankan setelah verifikasi token.

Baris 6-12 : Verifikasi token dengan mengirimkan token dan kunci yang diperlukan pada kode program parseToken.py kemudian

terima userId, email, dan role dalam bentuk *string* dari data yang telah dipisah menggunakan tanda "-" yang ditangkap menggunakan stdout.

Baris 17 : Lanjut ke fungsi berikutnya.

6. Logout

Source Code 4.10: Logout

```
1  router.get('/logout',
2    function(req, res) {
3     res.clearCookie('userData');
4     res.status(200).send({ auth: false, token: null });
5    }
6  );
```

Penjelasan *source code* 10 adalah sebagai berikut:

Baris 1 : Rute logout menggunakan get dan parameter /logout sebagai rute yang dihubungkan ke parameter kedua.

Baris 2 : Parameter function sebagai fungsi yang dijalankan untuk *logout*.

Baris 3 : Hapus nilai *cookie*.

Baris 4 : Server merespon HTTPP 200/Ok, autentikasi gagal dan token *null*

4.2.6 Pembuatan Token

Implementasi pembuatan token membutuhkan 2 kode program yaitu createKey untuk pembuatan kunci dan createToken menggunakan Python untuk pembuatan token. Kunci yang dibutuhkan ada 3 yaitu key untuk enkripsi/dekripsi, pkey untuk publik *key* untuk verifikasi token, dan *skey* untuk private *key* penghasil *signature*.

Source Code 4.11: Pembuatan kunci

```
1  pubkey, privkey = pysodium.crypto_sign_keypair()
2  mykey = secrets.token_bytes(32)
```

Penjelasan *source code* 11 adalah sebagai berikut:

Baris 1 : *Public key* dan *private* key untuk kebutuhan *signature* dihasilkan dengan menggunakan *library* Libsodium dengan ukuran masing-masing 32 bytes dan 64 bytes

Baris 2 : Kunci untuk kebutuhan enkripsi dan dekripsi dihasilkan dengan menggunakan *library* Secrets dengan ukuran 32 *bytes*.

Source Code 4.12: Pembuatan token

```
1  my_key = bytes.fromhex(sys.argv[4])
2  privkey = bytes.fromhex(sys.argv[5])
3  token = paseto.createCustom(
4   key=my_key,
5   skey=privkey,
6   purpose='custom',
7   claims={'my claims': [sys.argv[1], sys.argv[2],sys.argv[3]]},
8   exp_seconds=300
```

Penjelasan *source code* 12 adalah sebagai berikut:

Baris 1-2 :- Kunci diubah dari *hex* menjadi *bytes* yang berasal dari masukan kode program lain

Baris 3 : Token dihasilkan dengan memanggil fungsi createCustom dari kode program paseto dengan melampirkan beberapa parameter

Baris 4-8 : Parameter yang dilampirkan adalah kunci, *purpose* berupa custom, *claim* berupa user_id, email, dan *role* dari masukan kode program lain, dan masa berlaku sebuah token selama 300 detik

4.2.7 Penguraian Token

Implementasi pembuatan token membutuhkan kode program parseToken menggunakan Python untuk menguraikan token. Kunci yang dihasilkan sebelumnya juga ikut dilampirkan yaitu *key* untuk dekripsi dan pkey untuk publik *key* untuk verifikasi token.

Source Code 4.13: Penguraian token

```
1  tokenByte= bytes(sys.argv[1] , 'utf-8')
2  my_key = bytes.fromhex(sys.argv[2])
3  pubkey = bytes.fromhex(sys.argv[3])
4  try:
5   parsed = paseto.parseCustom(
6    key=my_key,
7    pkey=pubkey,
8    purpose='custom',
9    token=tokenByte,)
10  print (parsed['message']['my claims'][0]+'-'+parsed['message']['my
11 claims'][1]+'-'+parsed['message']['my claims'][2])
12 except:
13  print ("error")
```

Penjelasan *source code* 13 adalah sebagai berikut:

Baris 1 :- Token diubah dari *string* menjadi *bytes* yang berasal dari masukan kode program lain

Baris 2-3 : Kunci diubah dari *hex* menjadi *bytes* yang berasal dari masukan kode program lain

Baris 4-5 : Token diuraikan dengan memanggil fungsi createCustom dari kode program paseto dengan melampirkan beberapa parameter

Baris 5-9 : Parameter yang dilampirkan adalah kunci, *purpose* berupa custom token

Baris 10 : Cetak hasil penguraian token berupa userId, email, dan *role* yang dipisah menggunakan tanda "-"

Baris 12-13 : Cetak *error* jika terjadi kegagalan dalam penguraian token

4.2.8 PASETO

Implementasi PASETO dibutuhkan untuk membuat dan menguraikan sebuah token. Implementasi PASETO membutuhkan 9 kode program untuk pembuatan dan penguraian token menggunakan Python. Kemudian dibandingkan kembali apakah header adalah v2.local. Jika tidak, maka proses akan dihentikan.

1. *Pre-Authentication Encoding*

Source Code 4.14: Pre-Authentication Encoding

```
1  def pre_auth_encode(*parts):
2      accumulator = struct.pack('<Q', len(parts))
3      for part in parts:
4          accumulator += struct.pack('<Q', len(part))
5          accumulator += part
6      return accumulator
```

Penjelasan source code 14 adalah sebagai berikut:

Baris 1 :- *Instance method* yang digunakan untuk mengemas *header*, *nonce*/data, dan *footer* ke dalam bentuk *bytes*. Hasilnya digunakan sebagai metadata untuk mengecek keaslian sebuah data

Baris 2 : Menambahkan jumlah masukan yang diberikan pada *method* preAuth yang dikemas menggunakan *struct* dengan format *little endian* dan *unsigned integer*.

Baris 3-5 : Melakukan perulangan untuk mengemas panjang data dati setiap parameter yang diberikan diikuti dengan isi dari parameter tersebut dengan format *little endian* dan *unsigned integer*.

Baris 6 : Mengembalikan nilai accumulator sebagai hasil *pre-authentication encoding*

2. Enkode dan Dekode Base64

Source Code 4.15: Enkode dan Dekode Base64

```
1  def b64encode(data):
2      return base64.urlsafe_b64encode(data).rstrip(b'=')
3  def b64decode(data):
4      return base64.urlsafe_b64decode(data + b'=' * (-len(data) % 4))
```

Penjelasan source code 15 adalah sebagai berikut:

Baris 1-2 :- Encode data string menggunakan URL- dan filesystem-safe alphabet, Menggantikan + menjadi - dan / menjadi _ di standard Base64 alphabet. Hasilnya masih mengandung =. Kemudian mengembalikan data dalam bentuk bytes dengan menghilangkan b'='

Baris 3-4 : Mengembalikan data yang sudah diencode sebelumnya dengan menambah b'=' pada belakang data sebanyak panjang data sebelumnya

3. Json Encoder

Source Code 4.16: JsonEncoder

```
1  class JsonEncoder(object):
2    @classmethod
3    def dumps(cls, var):
4      return json.dumps(var, sort_keys=True, separators=(',',
5  ':')).encode('utf8')
6    @classmethod
7    def loads(cls, var):
8      return json.loads(var)
```

Penjelasan *source code* 16 adalah sebagai berikut:

Baris 1 :- *Class* untuk menghasilkan data bertipe json untuk *claim* yang akan digunakan.

Baris 2 : Menggunakan anotasi classmethod agar dapat dipanggil menggunakan *class* dan menggunakan variable *class*

Baris 3 : Fungsi yang digunakan untuk memasukkan data ke dalam json.

Baris 4-5 : Mengembalikan data dalam bentuk json dengan separator ‘,’ dan ‘:’ Dengan tipe enkode ‘utf-8’.

Baris 6 : Menggunakan anotasi classmethod agar dapat dipanggil menggunakan *class* dan menggunakan variable class

Baris 7 : Fungsi yang digunakan untuk mengeluarkan data dari json menjadi bentuk *dictionary*.

Baris 8 : Mengembalikan data dari json menjadi data dalam bentuk *dictionary*

4. Enkripsi

Source Code 4.17: Enkripsi

```
1  @classmethod
2  def encrypt(
3    cls,
4    plaintext: bytes,
5    key: bytes,
6    footer=b'',
7  ) -> bytes:
8    if cls.nonce_for_unit_testing:
9      nonce = cls.nonce_for_unit_testing
10     cls.nonce_for_unit_testing = None
11   else:
12     nonce = pysodium.randombytes(
13       pysodium.crypto_aead_xchacha20poly1305_ietf_NPUBBYTES)
14     nonce = pysodium.crypto_generichash(
15       plaintext,
16       k=nonce,
17       outlen=pysodium.crypto_aead_xchacha20poly1305_ietf_NPUBBYTES
18     )
19   ciphertext = pysodium.crypto_aead_xchacha20poly1305_ietf_encrypt(
20     message=plaintext,
21     ad=pre_auth_encode(cls.local_header, nonce, footer),
22     nonce=nonce,
23     key=key
```

Source Code 4.17: Enkripsi Lanjutan

```
24      )
25      token = cls.local_header + b64encode(nonce + ciphertext)
26      if footer:
27          token += b'.' + b64encode(footer)
28      return token
```

Penjelasan *source code* 17 adalah sebagai berikut:

Baris 1 :- Menggunakan anotasi classmethod agar dapat dipanggil menggunakan class dan menggunakan variable class

Baris 2-7 : Menggunakan function anotation untuk menghasilkan token berbentuk *bytes* dengan parameter cls, *plaintext bytes*, *key bytes*, dan *footer bytes*

Baris 8-10 : Periksa *nonce* yang digunakan apakah diproduksi sendiri atau sudah ditentukan untuk penggunaan tes

Baris 11-13 : Menghasilkan *random bytes* dari the OS's CSPRNG sebagai *key* ke dalam variabel *nonce* sepanjang 24 *bytes* ke dalam variabel *nonce*

Baris 14-18 : Menghasilkan *nonce* menggunakan *message plaintext* yang di-*hashing* menggunakan algoritme Blake2b sebagai *default hash* dengan *key* sepanjang 24 *bytes* pada variabel *nonce* yang sudah dihasilkan sebelumnya

Baris 19-24 : Menghasilkan *ciphertext* menggunakan *plaintext message*, hasil preAuth dari *header*, *nonce* dan *foorter*, yang dienkripsi menggunakan algoritme XChaCha20-Poly1305 dengan *nonce* dan *key*

Baris 25 : Menghasilkan token dengan menggabungkan *local header* dan hasil base64encode dari *nonce* dan *ciphertext*

Baris 26-27 : Tambahkan ke dalam token *footer* jika ada

Baris 28 : Mengembalikan token pada kode program yang memanggil *method* encrypt

5. Dekripsi

Source Code 4.18: Pembuatan Dekripsi

```
1   @classmethod
2   def decrypt(
3       cls,
4       token: bytes,
5       key: bytes
6   ) -> dict:
7       parts = token.split(b'.')
8       footer = b''
9       if len(parts) == 4:
10          encoded_footer = parts[-1]
11          footer = b64decode(encoded_footer)
```

Source Code 4.18: Pembuatan Dekripsi Lanjutan

```
12  header_len = len(cls.local_header)
13   header = token[:header_len]
14   token_version = token[:2]
15   if not secrets.compare_digest(token_version, cls.version):
16    raise InvalidVersionException('not a v2 token')
17   if not secrets.compare_digest(header, cls.local_header):
18    raise InvalidPurposeException('not a v2.local token')
19   decoded = b64decode(parts[2])
20   nonce = decoded[
21    :pysodium.crypto_aead_xchacha20poly1305_ietf_NPUBBYTES]
22   ciphertext = decoded[
23    pysodium.crypto_aead_xchacha20poly1305_ietf_NPUBBYTES:]
24   plaintext = pysodium.crypto_aead_xchacha20poly1305_ietf_decrypt(
25    ciphertext=ciphertext,
26    ad=pre_auth_encode(header, nonce, footer),
27    nonce=nonce,
28    key=key
29   )
30   return {
31    'message': plaintext,
32    'footer': footer if footer else None
33   }
```

Penjelasan *source code* 18 adalah sebagai berikut:

Baris 1 :- Menggunakan anotasi classmethod agar dapat dipanggil menggunakan class dan menggunakan variable class

Baris 2-6 : Menggunakan *function anotation* untuk menghasilkan pesan berbentuk *dictionary* dengan parameter cls, token *bytes*, dan *key bytes*

Baris 7 : Memisahkan antar komponen dengan '.' sebagai separator ke dalam variabel parts

Baris 8-11 : Mengekstrak *footer* dari token dengan mengambilnya dari indeks terakhir dari variabel parts kemudian dekode menggunakan base64

Baris 12-13 : Mendapatkan header dari variabel token dengan indeks sepanjang header_len , dimana panjang *header*_len sama dengan panjang local_header milik kelas.

Baris 14 : Dekode bagian *nonce* dan *ciphertext* pada variabel parts indeks ke-2 ke dalam variabel decoded

Baris 15-16 : Membandingkan versi token dari variabel parameter masukan dengan variabel milik kelas menggunakan secrets.compare_digest

Baris 17-18 : Membandingkan *header* token dari variabel parameter masukan dengan variabel milik kelas menggunakan secrets.compare_digest

Baris 19 : Dekode bagian nonce dan *ciphertext* pada variabel parts indeks ke-2 ke dalam variabel decoded

Baris 20-21 : Mendapatkan *nonce* dari dari 24 *bytes* awal dari variabel decoded

Baris 21-22 : Mendapatkan *ciphertext* dari variabel decode kecuali 24 *bytes* awal

Baris 24-29 : Dekripsi *ciphertext* menggunakan fungsi xhacha20poly1305_decrypt dari libssodium dengan parameter ciphertext, pre_auth_encode(header, nonce, footer), dan key ke dalam variabel *plaintext*

Baris 30-33 : Mengembalikan nilai *plaintext* ke dalam *dictionary* dengan indeks *message* dan *footer* jika ada

6. *Signature*

Source Code 4.19: Signature

```
1  @classmethod
2  def sign(cls, data, key, footer=b''):
3   signature = pysodium.crypto_sign_detached(
4    m=pre_auth_encode(cls.public_header, data, footer),
5    sk=key
6   )
7   token = cls.public_header + b64encode(data + signature)
8   if footer:
9    token += b'.' + b64encode(footer)
10  return token
```

Penjelasan *source code* 19 adalah sebagai berikut:

Baris 1 :- Menggunakan anotasi classmethod agar dapat dipanggil menggunakan class dan menggunakan variabel class

Baris 2 : Fungsi yang digunakan untuk menghasilkan *signature* dengan parameter data, *key*, dan *footer*

Baris 3-6 : Menghasilkan *signature* dari *message* dari hasil pre-authentication encoding *header*, *claim*, dan *footer* menggunakan algoritme Ed25519 dengan sk sebagai *secret key*

Baris 7 : Menghasilkan token dengan menggabungkan *public header* dan hasil b64encode dari gabungan data dan *signature*

Baris 8-9 : Tambahkan *footer* ke dalam token jika ada

Baris 10 : Mengembalikan token pada kode program yang memanggil method encrypt

7. Verifikasi

Source Code 4.20: Verifikasi

```
1  @classmethod
2  def verify(cls, token, key):
3   token_header = token[:len(cls.public_header)]
```

Source Code 4.20: Verifikasi Lanjutan

```
4   token_version = token[:2]
5   if not secrets.compare_digest(token_version, cls.version):
6    raise InvalidVersionException('not a v2 token')
7   if not secrets.compare_digest(token_header, cls.public_header):
8    raise InvalidPurposeException('not a v2.public token')
9   parts = token.split(b'.')
10  footer = b''
11  if len(parts) == 4:
12   encoded_footer = parts[-1]
13   footer = b64decode(encoded_footer)
14  decoded = b64decode(parts[2])
15  message = decoded[:-pysodium.crypto_sign_BYTES]
16  signature = decoded[-pysodium.crypto_sign_BYTES:]
17  try:
18   pysodium.crypto_sign_verify_detached(
19    sig=signature,
20    msg=pre_auth_encode(token_header, message, footer),
21    pk=key
22   )
23  except ValueError as e:
24   raise InvalidTokenException('invalid signature') from e
25  return {'message': message, 'footer': footer}
```

Penjelasan *source code* 20 adalah sebagai berikut:

Baris 1 :- Menggunakan anotasi classmethod agar dapat dipanggil menggunakan class dan menggunakan variable class

Baris 2 : Fungsi yang digunakan untuk memverifikasi token dengan parameter token dan *key*.

Baris 3 : Mendapatkan header dari variable token dengan indeks sepanjang header_len , dimana panjang header_len sama dengan panjang public_header milik kelas.

Baris 4 : De kode bagian nonce dan ciphertext pada variable parts indeks ke-2 ke dalam variable decoded

Baris 5-6 : Membandingkan versi token dari variable parameter masukan dengan variable milik kelas menggunakan secrets.compare_digest

Baris 7-8 : Membandingkan header token dari variable parameter masukan dengan variable milik kelas menggunakan secrets.compare_digest

Baris 9 : Memisahkan antar komponen dengan '.' sebagai separator ke dalam variable parts

Baris 10-13 : Mengekstrak footer dari token dengan mengambilnya dari indeks terakhir dari variable parts kemudian decode menggunakan base64

Baris 14 : Dekode bagian *nonce* dan ciphertext pada variable parts indeks ke-2 ke dalam variabel decoded

Baris 15 : Mendapatkan *message* dari variable decoded kecuali 64 bytes terakhir

Baris 16 : Mendapatkan *signature* dari dari 64 *bytes* terakhir dari variabel decoded

Baris 17-22 : Memverifikasi token dengan menggunakan fungsi crypto_sign_detached_verify dengan parameter *signature*, message dari hasil pre_auth_encode *header*, *message*, dan *footer* menggunakan kunci

Baris 23-24 : Jika *error*, maka *signature* bermasalah

Baris 25 : Mengembalikan nilai *plaintext* ke dalam *dictionary* dengan indeks *message* dan *footer* jika ada

8. Pembuatan Token

Source Code 4.21: Pembuatan token

```
1  def createCustom(
2   key,
3   skey,
4   purpose: str,
5   claims: dict,
6   exp_seconds=None,
7   footer=None,
8   encoder=JsonEncoder,
9  ):
10  if purpose not in {'custom'}:
11   raise InvalidPurposeException('invalid purpose')
12  if not key:
13   raise ValueError('key is required')
14  if not skey:
15   raise ValueError('secret key is required')
16  if exp_seconds:
17   then = pendulum.now().add(seconds=exp_seconds).to_atom_string()
18   claims['exp'] = then
19  encoded = encoder.dumps(claims)
20  encoded_footer = encoder.dumps(footer) if footer else b''
21  encrypt = PasetoV2.encrypt(
22   plaintext=encoded,
23   key=key,
24   footer=encoded_footer,
25  )
26  token = PasetoV2.sign(
27   data=encrypt,
28   key=skey,
29   footer=encoded_footer,
30  )
31  return token
```

Penjelasan source code 21 adalah sebagai berikut:

Baris 1 :- *Instance method* yang digunakan untuk menghasilkan token custom yang dihasilkan dengan cara mengenkripsi data kemudian ditandatangani

Baris 2 : Parameter key untuk kebutuhan enkripsi

Baris 3 : Parameter skey (secretKey) untuk kebutuhan *signature*

Baris 4 : Parameter purpose untuk kebutuhan *header* berupa *string*

Baris 5 : Parameter claim untuk data yang akan digunakan berupa dictionary

Baris 6 : Parameter exp untuk *rule expired token*

Baris 7 : Parameter footer jika dibutuhkan

Baris 8 : Parameter encoder untuk keperluan *load* dan *dump* data ke dalam json berupa JsonEncoder

Baris 10-11 : Cek *purpose*

Baris 12-13 : Cek *key* enkripsi

Baris 14-15 : Cek *key signature*

Baris 16-18 : Cek masa berlaku token, jika ada maka tambahkan ke dalam claim['exp'] dengan nilai jumlah waktu sekarang ditambah masa berlaku token dalam bentuk atom.string.

Baris 19 : Masukkan dictionary claim ke dalam json encoded

Baris 20 : Masukkan *footer* ke dalam encoded_footer

Baris 21-25 : Enkripsi pesan menggunakan method encrypt milik kelas PasetoV2 dengan parameter encoded, key, dan footer

Baris 26-30 : Pesan yang sudah dienkripsi kemudian ditambahkan *signature* menggunakan method sign milik kelasa PasetpV2 dengan parameter data berupa pesan terenkripsi, *key*, dan *footer* ke dalam variabel token

Baris 31 : Mengembalikan nilai token pada kode program yang memanggil method createCustom.

9. Penguraian Token

Source Code 4.22: Penguraian token

```
1  def parseCustom(
2    key,
3    pkey,
4    purpose: str,
5    token: bytes,
6    encoder=JsonEncoder,
7    validate: bool = True,
8    rules=None,
9    required_claims=None
10   ):
11   if purpose not in {'custom'}:
12     raise InvalidPurposeException('invalid purpose')
13   if not key:
14     raise ValueError('key is required')
15   if not pkey:
16     raise ValueError('public key is required')
17   resultSign = PasetoV2.verify(token, pkey)
18   result = PasetoV2.decrypt(resultSign['message'], key)
19   decoded_message = encoder.loads(result['message'])
20
```

```
21  decoded_footer = encoder.loads(result['footer']) if
22 result['footer'] else None
23  if required_claims:
24  missing_claims = set(required_claims).
25   difference(set(decoded_message.keys()))
26  if missing_claims:
27   raise PasetoValidationError(f'required claims missing
28 {missing_claims}')
29  rules = DEFAULT_RULES if not rules else set(rules)
30  if validate and not rules:
31   raise ValueError('must set validate=False to use no rules')
32  rule_set = {'exp'}
33  unknown_rules = rules.difference(rule_set)
34  if unknown_rules:
35   raise ValueError(f'unknown rules: {unknown_rules}')
36  if validate:
37   if 'exp' in rules and 'exp' in decoded_message:
38    exp = decoded_message['exp']
39    when = pendulum.parse(exp)
40    if pendulum.now() > when:
41    raise PasetoTokenExpired('token expired')
   return {'message': decoded_message, 'footer': decoded_footer}
```

Penjelasan *source code* 22 adalah sebagai berikut:

Baris 1 : *Instance method* yang digunakan untuk menguraikan token custom yang dengan cara memverifikasi token custom kemudian didekripsi.

Baris 2 : Parameter key untuk kebutuhan dekripsi

Baris 3 : Parameter pkey (publicKey) untuk kebutuhan verifikasi

Baris 4 : Parameter purpose untuk kebutuhan *header* berupa *string*

Baris 5 : Parameter encoder untuk keperluan *load* dan *dump* data ke dalam json berupa JsonEncoder

Baris 6 : Parameter validate berupa *boolean True*

Baris 7 : Parameter rule berupa None yang akan digunakan untuk validasi *rule* token

Baris 8 : Parameter required_claims yang digunakan untuk daftar *claim* yang akan digunakan

Baris 11-12 : Cek *purpose*

Baris 13-14 : Cek *key* dekripsi

Baris 15-16 : Cek *key* verifikasi

Baris 17 : Verifikasi token menggunakan method verify milik kelas PasetoV2 dengan parameter token dan pkey ke dalam variable result

Baris 18 : Dekripsi toke menggunakan method decrypt milik kelas PasetoV2 dengan parameter result dan key

Baris 19 : Ambil pesan dari json encoder menggunakan method load ke dalam variabel decoded_message

Baris 20-21 : Ambil footer dari json encoder menggunakan method load ke dalam variabel decoded_footer

Baris 22 : Cek jika ada daftar *claim* yang diwajibkan

Baris 23-27 : Cek jika ada *claim* yang hilang dari *claim* yang disyaratkan

Baris 28-34 : Cek rule yang diterapkan jika tidak ada maka *rules* yang dipakai exp

Baris 35-40 : Cek validate jika true maka validasi token berdasarkan *rule* seperti masa *expire* token

Baris 41 : Mengembalikan nilai *plaintext* ke dalam *dictionary* dengan indeks message dan footer jika ada

BAB 5 PENGUJIAN DAN ANALISIS

5.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk menguji 8 kebutuhan fungsional yang dimiliki oleh sistem ini. Pengujian ini dilakukan pada dua klien dengan *role* berbeda, yaitu admin dan klien biasa. Dimulai dari register/login, akses rute biasa, dan rute terlindungi hingga keluar dari sistem.

5.1.1 Tujuan Pengujian

Pengujian ini dilakukan untuk memastikan seluruh kebutuhan fungsional dapat berjalan tanpa adanya kesalahan. Terdapat lima kategori pengujian yang akan dilakukan. Lima kategori ini merepresentasikan rute-rute yang terdapat dalam sistem ini, yaitu :

1. POST /api/auth/register
2. POST /api/auth/login
3. GET /api/auth/me
4. GET /api/auth/allData
5. GET /api/auth/logout

5.1.2 Prosedur Pengujian

5.1.2.1 POST /api/auth/register

1. Klien dapat melakukan register
2. Server mengembalikan token dalam bentuk *cookies* bernama userData

5.1.2.2 POST /api/auth/login

1. Klien dapat melakukan *login*
2. Jika klien dengan *email* dan *password* yang dicari tidak ada, maka server mengembalikan status HTTP 404 dengan pesan "*No user found*"
3. Jika klien dengan *email* dan *password* yang dicari ada, maka server mengembalikan status HTTP 200 dengan pesan "auth: true, token: token"
4. Jika klien dengan *email* dan *password* yang dicari ada, maka server mengembalikan token dalam bentuk *cookies* bernama userData

5.1.2.3 POST /api/auth/me

1. Klien dapat mengakses rute me dengan mengirimkan *cookie* berisi token
2. Jika token yang dikirimkan kosong, maka server mengembalikan status HTTP 401 dengan pesan "auth: false, message: 'No token provided'"
3. Jika token tidak kosong namun tidak valid, maka server mengembalikan status HTTP 500 dengan pesan "*Token invalid*."

4. Jika token valid, maka server akan merespon dengan status HTTP 200 dan mengembalikan informasi mengenai klien itu sendiri

5.1.2.4 POST /api/auth/allData

1. Klien dapat mengakses rute allData dengan mengirimkan *cookie* berisi token
2. Jika token yang dikirimkan kosong, maka server mengembalikan status HTTP 401 dengan pesan "auth: false, message: 'No token provided'"
3. Jika token tidak kosong namun tidak valid, maka server mengembalikan status HTTP 500 dengan pesan "*Token invalid*"
4. Jika token valid namun *role* bukan admin, maka server akan merespon dengan status HTTP 401 dengan pesan "*You need administrator permission to access this route*"
5. Jika token valid dan role admin, maka server akan merespon dengan status HTTP 200 dan mengembalikan informasi mengenai user itu sendiri

5.1.2.5 POST /api/auth/logout

1. Menghapus token dalam cookie
2. Ketika cookie sudah dihapus server merespon dengan status 200 dan pesan "auth: false, token: null"

5.1.3 Hasil dan Analisis

Tabel 5.1 Pengujian POST /api/auth/register

Nama kasus uji	POST /api/auth/register
Prosedur	Klien memasukkan data nama, email, *password*, dan *role*
Hasil yang diharapkan	1. Server mengembalikan status HTTP 200 dengan pesan "auth: true, token: token" 2. Server mengembalikan token dalam bentuk *cookies* bernama userData
Hasil	1. Server mengembalikan status HTTP 200 dengan pesan "auth: true, token: token" 2. Server mengembalikan token dalam bentuk *cookies* bernama userData
Status	Valid

Tabel 5.2 Pengujian POST /api/auth/login

Nama kasus uji	POST /api/auth/login
Prosedur	Klien memasukkan data *email* dan *password*
Hasil yang diharapkan	1. Jika klien dengan email dan password yang dicari tidak ada, maka server mengembalikan status HTTP 404 dengan pesan "*No user found*" 2. JIka klien dengan email dan password yang dicari ada, maka server mengembalikan status HTTP 200 dengan pesan "auth: true, token: token" 3. Jika klien ada, maka server mengembalikan token dalam bentuk *cookies* bernama userData
Hasil	1. Server mengembalikan status HTTP 404 dengan pesan "*No user found*" saat klien dengan *email* dan *password* yang dicari tidak ada 2. Server mengembalikan status HTTP 200 dengan pesan "auth: true, token: token" saat masukan valid dan klien dengan *email* dan *password* yang dicari ada 3. Server mengembalikan token dalam bentuk cookies bernama userData saat klien dengan *email* dan *password* yang dicari ada
Status	Valid

Tabel 5.3 Pengujian GET /api/auth/me

Nama kasus uji	GET /api/auth/me
Prosedur	Klien mengirimkan token dalam bentuk *cookie* saat mengakses rute me
Hasil yang diharapkan	1. Klien dapat mengakses rute me dengan mengirimkan *cookie* berisi token

	2. Jika token yang dikirimkan kosong, maka server mengembalikan status HTTP 401 dengan pesan *"auth: false, message: 'No token provided'"* 3. JIka token tidak kosong namun tidak valid, maka server mengembalikan status HTTP 500 dengan pesan "*Token invalid*." 4. Jika token valid, maka server akan merespon dengan status HTTP 200 dan mengembalikan informasi mengenai klien itu sendiri
Hasil	1. Klien dapat mengakses rute me dengan mengirimkan *cookie* berisi token 2. Server mengembalikan status HTTP 401 dengan pesan *"auth: false, message: 'No token provided'"* saat token yang dikirimkan kosong 3. Server mengembalikan status HTTP 500 dengan pesan "*Token invalid*" saat token tidak kosong namun tidak valid 4. Server akan merespon dengan status 200 dan mengembalikan informasi mengenai klien itu sendiri saat token valid
Status	Valid

Tabel 5.4 Pengujian GET /api/auth/allData

Nama kasus uji	GET /api/auth/allData
Prosedur	Klien mengirimkan token dalam bentuk *cookie* saat mengakses rute me
Hasil yang diharapkan	1. Klien dapat mengakses rute allData dengan mengirimkan *cookie* berisi token 2. Jika token yang dikirimkan kosong, maka server mengembalikan status HTTP 401 dengan pesan *"auth:*

	false, message: 'No token provided'" 3. Jika token tidak kosong namun tidak valid, maka server mengembalikan status HTTP 500 dengan pesan "*Token invalid*" 4. Jika token valid namun role bukan admin, maka server akan merespon dengan status 401 dengan pesan *"You need administrator permission to access this route"* 5. Jika token valid dan *role* admin, maka server akan merespon dengan status HTTP 200 dan mengembalikan informasi seluruh klien
Hasil	1. Klien dapat mengakses rute allData dengan mengirimkan *cookie* berisi token 2. Server mengembalikan status htpp 401 dengan pesan *"auth: false, message: 'No token provided'"* saat token yang dikirimkan kosong 3. Server mengembalikan status HTTP 500 dengan pesan "*Token invalid*" saat token tidak kosong namun tidak valid 4. Server akan merespon dengan status HTTP 401 dengan pesan *"You need administrator permission to access this route"* saat token valid namun *role* bukan admin 5. Server akan merespon dengan status HTTP 200 dan mengembalikan informasi seluruh klien saat token valid dan *role* admin
Status	Valid

Tabel 5.5 Pengujian GET /api/auth/logout

Nama kasus uji	POST /api/auth/logout
Prosedur	Klien mengirimkan token dalam bentuk *cookie* saat mengakses rute me
Hasil yang diharapkan	1. Menghapus token dalam *cookie* 2. Ketika *cookie* sudah dihapus server merespon dengan status HTTP 200 dan pesan "*auth: false, token: null*"
Hasil	1. Cookie kosong 2. Server merespon dengan status HTTP 200 dan pesan "*auth: false, token: null*"
Status	Valid

5.2 Pengujian Nonfungsional

5.2.1 Tujuan Pengujian

Pengujian ini dilakukan untuk memastikan seluruh kebutuhan nonfungsional dapat berjalan tanpa adanya kesalahan.

5.2.2 Prosedur Pengujian

Pengujian ini dilakukan saat klien mencoba menggunakan rute POST api/auth/register atau POST api/auth/login. Saat rute itu diakses, maka server akan mencoba melakukan *hashing* pada *password* yang dikirimkan. *Hashing password* yang digunakan adalah Bcrypt.

5.2.3 Hasil dan Analisis

Tabel 5.6 *Hashing password*

Nama kasus uji	*Hashing password*
Prosedur	Klien mengakses rute POST api/auth/register atau POST api/auth/login.
Hasil yang diharapkan	Server melakukan *hashing* pada *password*
Hasil	Password yang dimasukkan sudah di-*hashing*
Status	Valid

5.3 Pengujian Keamanan

Pengujian keamanan dilakukan dengan mencoba menyerang PASETO menggunakan jenis serangan yang sama yang dilakukan pada JWT. Serangan yang dilakukan ada dua yaitu ekspoitasi algoritme None dan ekploitasi algoritme simetris-asimetris. Untuk pembahasan yang lebih menyeluruh, berikut ini detail penyerangan yang dilakukan.

5.3.1 Tujuan Pengujian

Pengujian ini dilakukan untuk mengetahui apakah PASETO mampu menangani masalah yang dialami oleh JWT. Masalah yang dialami yaitu pemalsuan token. Untuk itu diharapkan PASETO dapat mengidentifikasi token yang asli dan yang palsu. Selain itu juga dilakukan pengecekan apakah *payload* yang dihasilkan oleh PASETO Custom dapat didekode dan dapat dibaca dengan jelas

5.3.2 Prosedur Pengujian

5.3.2.1 Base64 Enkode/Dekode

1. Ambil token PASETO Custom
2. Dekode menggunakan Base64URL
3. Dekode menggunakan Base64URL Safe

5.3.2.2 Eksploitasi Algoritme None

1. Pemalsuan token dilakukan pada server yang mengharapkan token dengan *purpose custom*
2. Token didapatkan dari *cookie*
3. Bagian *signature* dihilangkan dari token
4. Pemalsuan token dilakukan dengan mengubah *header* menjadi *None*.
5. *Claim* yang dimiliki token hanya dienkode saja tanpa ada tambahan enkripsi dan *signature*.

5.3.2.3 Eksploitasi Algoritme Simetrik-Asimetrik

1. Pemalsuan token dilakukan pada server yang mengharapkan token dengan *purpose custom*
2. Token didapatkan dari *cookie*
3. Bagian *signature* dihilangkan dari token
4. Pemalsuan token dilakukan dengan mengubah *header public* menjadi *local*
5. Token dipalsukan dengan menghasilkan *signature* dari PASETO dengan *purpose local* menggunakan *public key* dari PASETO dengan *purpose public*
6. *Signature* yang dihasilkan ditambahkan pada token palsu

5.3.3 Hasil dan Analisis

Tabel 5.7 Base64 Enkode/Dekode

Nama kasus uji	Eksploitasi Algoritme None
Prosedur	1. Ambil token Paseto Custom 2. Dekode menggunakan Base64URL 3. Dekode menggunakan Base64URL Safe
Token	v2.local.djlubG9jYWwuaHF2ZnFSSEs2TmtLSWFuNW5oaEZRcW5XLXBmQlR2ZGt4RkNsQXFRV3lGNFd5ODdtdWtJc2lGdVhDeXdWMk9nU3p2SGRDem1KZHNHSkRSRHdBTjBNb04tWHEtcUxNbURlR3pqN2QzNi01QUppbS1HejgzMk14ZGZHb2ZIWFV6TkNOTlJOZXdUck9KN1ZFVHl1a2VRcHpNa1F0RWFZMXVkWlMo7D7yB67IwTHllC1tRSEoWhkkRgTQrCoPWKOwzUMd7QfYSqfbOA6lgc7MuftaV90j65T_H9FINvT2F8ABiQ0
Hasil yang diharapkan	Token tidak terbaca dengan jelas
Hasil	1. Base64URL: v2.local.hqvfqRHK6NkKIan5nhhFQqnW-pfBTvdkxFClAqQWyF4Wy87mukIsiFuXCywV2OgSzvHdCzmJdsGJDRDwAN0MoN-Xq-qLMmDeGzj7d36-5AJim-Gz832MxdfGofHXUzNCNNRNewTrOJ7VETyukeQpzMkQtEaY1udZS(>[illegible]1[illegible]- mE!(Z$FЬ*X[illegible]C[illegible]J[illegible]8[illegible] [illegible]ZW[illegible]H6[illegible] 2. v2.local.hqvfqRHK6NkKIan5nhhFQqnW-pfBTvdkxFClAqQWyF4Wy87mukIsiFuXCywV2OgSzvHdCzmJdsGJDRDwAN0MoN-Xq-qLMmDeGzj7d36-5AJim-Gz832MxdfGofHXUzNCNNRNewTrOJ7VETyukeQpzMkQtEaY1udZS(\xe

	c>\xf2\x07\xae\xc8\xc11\xe5\x94-mE!(Z\x19$F\x04\xd0\xac*\x0fX\xa3\xb0\xcdC\x1d\xed\x07\xd8J\xa7\xdb8\x0e\xa5\x81\xce\xcc\xb9\xfbZW\xdd#\xeb\x94\xff\x1f\xd1H6\xf4\xf6\x17\xc0\x01\x89\r'
Status	Valid

Pada tabel 6.7 terlihat *payload* yang didekode menggunakan Base64URL menghasilkan data masih dalam bentuk enkripsi sedangkan *signature* berisi karakter tidak jelas. Sedangkan saat didekode menggunakan Base64URLSafe, *payload* yang dihasilkan sudah terlihat jelas namun tidak dibaca. Hal ini dikarenakan data pada token masih dalam bentuk enkripsi. Untuk mengenkripsi sebuah data dibutuhkan kunci. Akan tetapi *key* yang dibutuhkan bukanlah *public*, sehingga hanya server yang mengetahui kuncinya.

Tabel 5.8 Eksploitasi Algoritme None

Nama kasus uji	Eksploitasi Algoritme None
Prosedur	1. Pemalsuan token dilakukan pada server yang mengharapkan token dengan *purpose public* 2. Token didapatkan dari *cookie* 3. Bagian *signature* dihilangkan dari token 4. Pemalsuan token dilakukan dengan mengubah *header* menjadi None. 4. *Claim* yang dimiliki token hanya dienkode saja tanpa ada tambahan enkripsi dan *signature*
Hasil yang diharapkan	PASETO merespon *error* dengan pesan *"invalid purpose"*
Hasil	PASETO merespon *error* dengan pesan *"invalid purpose"*
Status	Valid

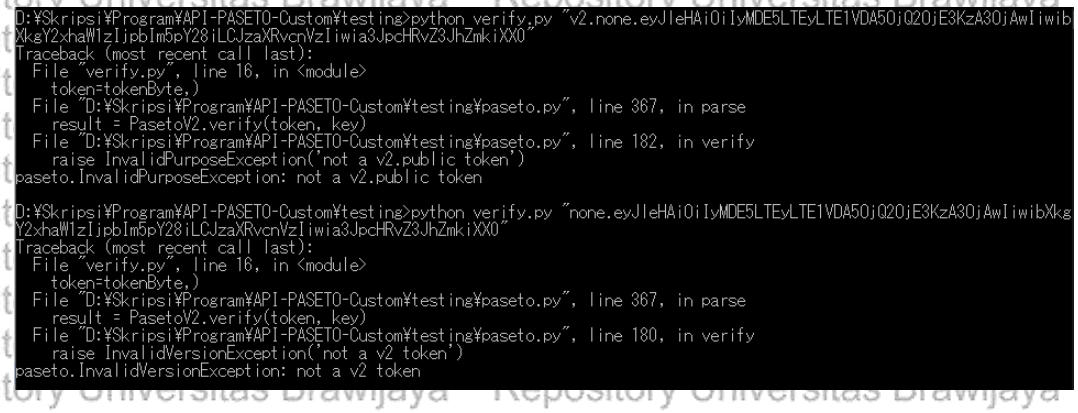
Pada tabel 6.8 dilakukan pengujian eksploitasi algoritme None pada PASETO dengan versi v2.public. V2.public digunakan karena hanya versi ini yang menggunakan *signature* saja tanpa enkripsi. Sehingga data dalam token dapat diketahui.

Gambar 5.1 Token v2.public

Pada gambar 6.1 terlihat sebuah token dengan data "nico", "Sitorus", dan "kriptografi" dibuat dengan menggunakan PASETO v2.public. Token ini akan digunakan sebagai dasar pemalsuan token menggunakan algoritme None. Token kemudian dipisah menjadi *header* dan *payload*. *Payload* yang sudah dipisah akan didekode menggunakan Base64URL Safe.

Gambar 5.2 Enkode Payload

Pada gambar 6.2 terlihat *payload* yang sudah didekode. Data dalam token terdiri atas masa berlaku token, "nico", "Sitorus", dan kriptografi". Kemudian data ini akan didekode kembali tanpa adanya *signature*.



Gambar 5.3 Verifikasi Token

Pada gambar 6.3 dilakukan verifikasi pada token yang sudah dipalsukan dengan *header* "v2.none". Hasil verifikasi memperlihatkan adanya *error* dengan pesan "paseto.InvalidPurposeException: not a v2.public *token*". Artinya *token* yang diverifikasi memiliki *purpose header* yang tidak valid. Token yang dipalsukan dengan *header* "none" juga memperlihatkan adanya *error* dengan pesan "paseto.InvalidVersionException: not a v2 token". Artinya token yang diverifikasi memiliki versi yang tidak valid.

Tabel 5.9 Eksploitasi Algoritme Simetris-Asimetris

Nama kasus uji	Eksploitasi Algoritme Simetris-Asimetris

Prosedur	1. Pemalsuan token dilakukan pada server yang mengharapkan token dengan *purpose custom* 2. Token didapatkan dari cookie 3. Bagian *signature* dihilangkan dari token 4. Pemalsuan token dilakukan dengan mengubah *header public* menjadi *local* 5. Token dipalsukan dengan menghasilkan *signature* dari PASETO dengan *purpose local* menggunakan *public key* dari PASETO dengan *purpose public* 6. *Signature* yang dihasilkan ditambahkan pada token palsu
Hasil yang diharapkan	PASETO merespon *error* dengan pesan "*invalid purpose*"
Hasil	PASETO merespon *error* dengan pesan "*invalid purpose*"
Status	Valid

Pada tabel 6.8 dilakukan pengujian eksploitasi algoritme simetris-asimetris pada PASETO dengan versi v2.public dan v2.local. V2.public dan v2.local digunakan karena hanya versi ini yang menggunakan algoritme asimetris dan simetris. Pemalsuan digunakan dengan mengganti *header* dari *public* menjadi *local* dan *signature* dihasilkan menggunakan *public key* dari v2.public.

```
D:¥Skripsi¥Program¥API-PASETO-Custom¥testing>python sign.py "nico" "sitorus" "kriptografi"
v2.public.eyJleHAiOiIyMDE5LTEyLTE1VDA5OjQ2OjE3KzA3OjAwIiwibXkgY2xhaWIzIjpbIm5pY28iLCJzaXRvcnVzIiwia3JpcHRvZ3JhZmkiXX0XpU
-mtn5vpHzr4cWivkQIoWwsRJ0fgIDNx8kmCTwb07LdjIkT3CGN6A1EODUtDMGma47h-01IP2do0JOHjHkK
```

Gambar 5.4 Token v2.public

Pada gambar 6.4 terlihat sebuah token dengan data "nico", "Sitorus", dan "kriptografi" dibuat dengan menggunakan PASETO v2.public. Token ini akan digunakan sebagai dasar pemalsuan token simetris-asimetris. Token kemudian dipisah menjadi *header* dan *payload*. *Payload* yang sudah dipisah akan didekode menggunakan Base64URL Safe.

Gambar 5.5 Enkode Payload

Pada gambar 6.5 terlihat *payload* yang sudah didekode. Data dalam token terdiri atas masa berlaku token, "nico", "Sitorus", dan kriptografi".

Gambar 5.6 Enkripsi

Enkripsi dilakukan dengan menggunakan *public* key yang digunakan pada pembuatan token PASETO dengan header v2.public.

Gambar 5.7 Verifikasi token

Pada gambar 6.7 dilakukan verifikasi pada token yang sudah dipalsukan dengan *header* "v2.local". Hasil verifikasi memperlihatkan adanya *error* dengan pesan "paseto.InvalidPurposeException: not a v2.public token". Artinya token yang diverifikasi memiliki *purpose header* yang tidak valid.

5.4 Pengujian Performa

Pada pengujian performa akan dilakukan 2 tipe pengujian yaitu pengujian pembuatan token dan pengujian penguraian token dengan mempertimbangkan waktu. Waktu yang digunakan untuk pembuatan token dan penguraian token yang dilakukan pada PASETO akan dibandingkan dengan JWT.

5.4.1 Tujuan Pengujian

Pengujian ini dilakukan untuk mengetahui rata-rata waktu yang dibutuhkan PASETO untuk membuat dan menguraikan sebuah token dari *claim* yang dikirimkan. Pengujian ini juga dilakukan untuk membandingkan kemampuan PASETO dan JWT dari segi waktu yang dibutuhkan untuk pemrosesan sebuah token.

5.4.2 Prosedur Pengujian

5.4.2.1 Pembuatan Token

1. JWT yang dibandingkan adalah JWT dengan algoritme HS256 dan RS256
2. PASETO yang digunakan adalah PASETO Custom dengan algoritme Blake2B untuk menghasilkan *nonce*, xChachaPoly1305 untuk enkripsi dan algoritme Ed25519 untuk *signature*

3. Ukuran *key* yang dibutuhkan oleh JWT untuk algoritme HS256 32 *bytes*
4. Ukuran *key* yang dibutuhkan oleh JWT untuk algoritme RSA265 adalah 64 *bytes*
5. Ukuran *key* yang dibutuhkan oleh PASETO untuk algoritme Blake2B adalah 24 *bytes* untuk menghasilkan *nonce*
6. Ukuran *key* yang dibutuhkan oleh PASETO untuk algoritme xChachaPoly1305 adalah 32 *bytes*
7. Ukuran *key* yang dibutuhkan oleh PASETO untuk algoritme Ed25519 adalah 64 *bytes*
8. *Key* yang digunakan untuk setiap kali pengujian sama
9. Pengujian akan dilakukan sebanyak 20 puluh kali untuk mengetahui rata-rata waktu pemrosesan.

5.4.2.2 Penguraian Token

1. JWT yang dibandingkan adalah JWT dengan algoritme HS256 dan RS256
2. PASETO yang digunakan adalah PASETO Custom dengan xChachaPoly1305 untuk dekripsi dan algoritme Ed25519 untuk verifikasi
3. Ukuran *key* yang dibutuhkan oleh JWT untuk algoritme HS256 32 *bytes*
4. Ukuran *key* yang dibutuhkan oleh JWT untuk algoritme RSA265 adalah 32 *bytes*
5. Ukuran *key* yang dibutuhkan oleh PASETO untuk algoritme xChachaPoly1305 adalah 32 *bytes*
6. Ukuran *key* yang dibutuhkan oleh PASETO untuk algoritme Ed25519 adalah 32 *bytes*
7. *Key* yang digunakan untuk setiap kali pengujian sama
8. Pengujian akan dilakukan sebanyak 20 puluh kali untuk mengetahui rata-rata waktu pemrosesan.

5.4.3 Hasil dan Analisis

Tabel 5.10 Waktu yang dibutuhkan untuk pembuatan token (ms)

Pengujian	PASETO v2	JWT	
	Custom	HS256	RSA256
1	2,6775	15,0403	8,3509
2	0,2482	0,2113	0,4221
3	0,2347	0,1052	0,3092
4	0,2206	0,1548	0,3575
5	0,3338	0,1314	0,3468
6	0,4197	0,1116	0,3799

7	0,2655	0,2144	0,3414
8	0,2042	0,1200	0,3265
9	0,2066	0,1318	0,3310
10	0,4366	0,1136	0,3205
11	0,2419	0,1041	0,3166
12	0,2285	0,1051	0,4385
13	0,2258	0,1062	0,3276
14	0,2140	0,1024	0,3441
15	0,2958	0,1348	0,3045
16	0,2263	0,1143	0,3013
17	0,2063	0,9503	1,1535
18	0,2012	0,1109	0,3091
19	0,3748	0,1663	0,3158
20	0,3726	0,1208	0,3089
21	0,2419	0,1220	0,3287
22	0,2285	0,1248	0,3113
23	0,2258	0,1328	0,3126
24	0,214	0,1271	0,3100
25	0,2958	0,1009	0,3740
26	0,2263	0,1005	0,3193
27	0,2063	0,1007	0,3042
28	0,2012	0,1754	0,3345
29	0,3748	0,1362	0,3003
30	0,3726	0,1206	0,3354
Rata-rata	0,347393	0,65302	0.627867

Pada tabel 6.10 dapat dilihat bahwa waktu rata-rata yang diperlukan oleh PASETO Custom untuk membuat token adalah 0,347393 ms. Lebih cepat dibanding HS256 yang membutuhkan waktu rata-rata 0,65302 ms. PASETO juga lebih cepat dibanding RSA256 yang membutuhkan waktu rata-rata 0.627867 ms. PASETO memiliki selisih waktu 0,305627 ms dibanding HSA256 dan 0,280474 ms dibanding RS256.

Tabel 5.11 Waktu yang dibutuhkan untuk penguraian token (ms)

Pengujian	PASETO v2	JWT	
	Custom	HS256	RSA256
1	0,3497	2,2377	12,4124
2	0,2512	0,1323	0,1763
3	0,2432	0,1817	0,1584
4	0,2866	0,1180	0,1609
5	0,2596	0,1076	0,2187
6	0,3457	0,1122	0,1743
7	0,2382	0,1196	0,1489
8	0,2402	0,115	0,1484
9	0,2348	0,1134	0,1643
10	0,2755	0,1015	0,1611
11	0,2720	0,0954	0,1456
12	0,2320	0,0962	0,1744
13	0,2683	0,0942	0,1573
14	0,3354	0,2110	0,1558
15	0,2515	0,1129	0,1480
16	0,2378	0,1070	0,1777
17	0,2313	0,1096	0,1484
18	0,5122	0,1111	0,1619
19	0,4661	0,1216	0,1452
20	0,3563	0,1157	0,2695
21	0,2720	0,1075	0,1393
22	0,2320	0,1044	0,1287
23	0,2683	0,1242	0,1484
24	0,3354	0,1005	0,1395
25	0,2515	0,0941	0,1622
26	0,2378	0,093301	0,1386
27	0,2313	0,0936	0,1339
28	0,5122	0,109499	0,1438

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

29	0,4661	0,111999	0,1718
30	0,3563	0,1076	0,1468
Rata-rata	0,301683	0.185347	0.568683

Pada tabel 6.11 dapat dilihat bahwa waktu rata-rata yang diperlukan oleh PASETO Custom untuk membuat token adalah 0,301683 ms. Lebih lambat dibanding HS256 yang membutuhkan waktu rata-rata 0.185347 ms. PASETO lebih cepat dibanding RSA256 yang membutuhkan waktu rata-rata 0.568683 ms. PASETO memiliki selisih waktu 0.116336 ms dibanding HSA256 dan 0,267 ms dibanding RS256.

BAB 6 PENUTUP

6.1 Kesimpulan

1. PASETO dapat diterapkan sebagai mekanisme autentikasi pada RESTful API menggunakan *wrapper* Pysodium agar dapat menggunakan *library* Libsodium berbasis bahasa pemrograman C. Kemudian PASETO dapat dipanggil menggunakan *child process* agar dapat diterapkan pada RESTful API berbasis nodeJS. Kemudian PASETO dimodifikasi agar setiap *payload* token dienkripsi menggunakan xChachaPoly1305 dan ditandatangani menggunakan Ed25519. Dengan penerapan tersebut, Untuk masuk ke dalam sistem, maka klien harus mendaftarkan nama, *email*, *password*, dan *role* yang akan digunakan untuk keperluan autentikasi. *Email* dan *password* untuk mengidentifikasi apakah user sudah terdaftar pada sistem. Sedangkan nama, *email*, dan role digunakan untuk pembuatan token. Token akan digunakan untuk mengkonfirmasi rute-rute mana saja yang dapat dimasuki oleh klien. Sehingga Ketika klien tidak berwenang atau tidak memiliki token, maka klien tidak dapat memasuki rute tersebut.
2. PASETO Custom untuk membuat token adalah 0,347393 ms. Lebih cepat dibanding HS256 yang membutuhkan waktu rata-rata 0,65302 ms. PASETO juga lebih cepat dibanding RSA256 yang membutuhkan waktu rata-rata 0.627867 ms. PASETO memiliki selisih waktu 0,305627 ms dibanding HSA256 dan 0,280474 ms dibanding RS256. PASETO Custom untuk membuat token adalah 0,301683 ms. Lebih lambat dibanding HS256 yang membutuhkan waktu rata-rata 0.185347 ms. PASETO lebih cepat dibanding RSA256 yang membutuhkan waktu rata-rata 0.568683 ms. PASETO memiliki selisih waktu 0.116336 ms dibanding HSA256 dan 0,267 ms dibanding RS256.

6.2 Saran

1. PASETO sebaiknya diimplementasikan menggunakan bahasa yang sama dengan server RESTFul API agar setiap *raise exception* yang ada pada paseto dapat dibaca jika dibutuhkan untuk kepentingan debug.
2. Skema *user* pada mongoDB sebaiknya ditambahkan konfigurasi pada email, sehingga setiap *email* yang didaftarkan unik.

DAFTAR PUSTAKA

Arciszewski, Scott. 2017. *No Way, JOSE! Javascript Object Signing and Encryption is a Bad Standard That Everyone Should Avoid*. Tersedia di: < https://paragonie.com/blog/2017/03/jwt-json-web-tokens-is-bad-standard-that-everyone-should-avoid/> [Diakses 2 Agustus 2018]

Arciszewski, Scott. 2018. *PASETO: Platform-Agnostic SEcurity Tokens*. Internet Engineering Task Force. Tersedia di: < https://tools.ietf.org/html/draft-paragon-paseto-rfc-00> [Diakses 2 Agustus 2018]

Bloch, Josua. 2006. *How to Design a Good API and Why It Matters*. dalam: Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA '06. ACM New York: New York, pp. 506-507.

Bradley, John dan Nat, Sakimura. 2015. *JSON Web Token*. Internet Engineering Task Force. Tersedia di: <https://tools.ietf.org/html/rfc7519> [Diakses 2 Agustus 2018]

Libsodium. 2013. *The Sodium Crypto Library (Libsodium)*. Tersedia di: <https://libsodium.gitbook.io/doc/> [Diakses 2 Agustus 2018]

NaCl. 2013. NaCl: *Networking and Cryptography Library.* Tersedia di: <nacl.cr.yp.to> [Diakses 2 Agustus 2018]

Emery, David. 1994. *Standards, APIs, Interfaces and Bindings*. Tersedia di:< https://web.archive.org/web/20150116081559/http://www.acm.org/tsc/apis.html> [Diakses 20 Juni 2019]

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

Jones, M. 2015. *JSON Web Token (JWT)*. Tersedia di: <https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-25> [Diakses 20 Agustus 2018]

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

Langkemper, Sjoerd. 2016. *Attacking JWT authentication*. Tersedia di: < https://www.sjoerdlangkemper.nl/2016/09/28/attacking-jwt-authentication/> [Diakses 1 Agustus 2018]

McLean, Tim. 2015. Critical Vulnerabilities in JSON Web Token Libraries. Tersedia di: <https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/> [Diakses 20 Juni 2019]

Richardson, Leonard dan Ruby, Sam. 2007. *RESTful Web Services*. California: O'Reilly Media, Inc, pp: 96-97.

Saarinen , Markku-Juhani O. dan Aumasson , Jean-Philippe. 2015. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. Request for Comments. Tersedia di: <https://www.rfc-editor.org/rfc/rfc7693.txt> [Diakses 2 Agustus 2018]

Satria, Bagus, dkk. 2018. *Implementasi Algoritme Blake2s pada JSON Web Token (JWT) sebagai Algoritme Hashing untuk Mekanisme Autentikasi Layanan REST-API*. S1. Tersedia di: < http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/3574/1410> [Diakses 1 Agustus 2018]

LAMPIRAN A PERBANDINGAN KINERJA PASETO DAN JWT DALAM PEMBUATAN TOKEN

A.1 Pembuatan Token

Pada tabel 6.1, dilampirkan perbandingan kinerja PASETO baik v2 custom, v2.public, dan v2.local maupun JWR HS256 dan JWT RS256 berdasarkan waktu yang dibutuhkan untuk membuat token.

Tabel 6.1 Waktu Yang Dibutuhkan Untuk Pembuatan Token (ms)

Pengujian	PASETO			JWT	
	PASETO v2 Custom	PASETO v2,local	PASETO v2,public	HS256	RSA256
1	2,6775	1.9575	2.493	15,0403	8,3509
2	0,2482	0.133	0.1812	0,2113	0,4221
3	0,2347	0.1295	0.1749	0,1052	0,3092
4	0,2206	0.1305	0.2017	0,1548	0,3575
5	0,3338	0.1567	0.1996	0,1314	0,3468
6	0,4197	0.1494	0.2046	0,1116	0,3799
7	0,2655	0.1375	0.1988	0,2144	0,3414
8	0,2042	0.1354	0.1849	0,1200	0,3265
9	0,2066	0.1347	0.18	0,1318	0,3310
10	0,4366	0.1326	0.1783	0,1136	0,3205
11	0,2419	0.1341	0.1775	0,1041	0,3166
12	0,2285	0.179	0.1921	0,1051	0,4385
13	0,2258	0.2056	0.191	0,1062	0,3276
14	0,2140	0.1372	0.3108	0,1024	0,3441
15	0,2958	0.1329	0.2039	0,1348	0,3045
16	0,2263	0.13	0.1915	0,1143	0,3013
17	0,2063	0.1483	0.1913	0,9503	1,1535
18	0,2012	0.3241	0.1795	0,1109	0,3091
19	0,3748	0.1994	0.1765	0,1663	0,3158
20	0,3726	0.1374	0.1747	0,1208	0,3089
21	0,2419	0.1337	0.2255	0,1220	0,3287

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

22	0,2285	0.1354	0.2835	0,1248	0,3113
23	0,2258	0.1321	0.1833	0,1328	0,3126
24	0,214	0.1321	0.1783	0,1271	0,3100
25	0,2958	0.1321	0.1906	0,1009	0,3740
26	0,2263	0.1322	0.1823	0,1005	0,3193
27	0,2063	0.1681	0.1812	0,1007	0,3042
28	0,2012	0.2563	0.1795	0,1754	0,3345
29	0,3748	0.1485	0.188	0,1362	0,3003
30	0,3726	0.1367	0.2372	0,1206	0,3354
Rata-rata	0,347393	0.21436	0.27384	0,65302	0.627867

A.2 Penguraian Token

Pada tabel 6.2, dilampirkan perbandingan kinerja PASETO baik v2 custom, v2.public, dan v2.local maupun JWR HS256 dan JWT RS256 berdasarkan waktu yang dibutuhkan untuk menguraikan token.

Tabel 6.2 Waktu Yang Dibutuhkan Untuk Penguraian Token (ms)

Pengujian	PASETO			JWT	
	PASETO v2 Custom	PASETO v2,local	PASETO v2,public	HS256	RSA256
1	2,6775	1.9575	2.493	15,0403	8,3509
2	0,2482	0.133	0.1812	0,2113	0,4221
3	0,2347	0.1295	0.1749	0,1052	0,3092
4	0,2206	0.1305	0.2017	0,1548	0,3575
5	0,3338	0.1567	0.1996	0,1314	0,3468
6	0,4197	0.1494	0.2046	0,1116	0,3799
7	0,2655	0.1375	0.1988	0,2144	0,3414
8	0,2042	0.1354	0.1849	0,1200	0,3265
9	0,2066	0.1347	0.18	0,1318	0,3310
10	0,4366	0.1326	0.1783	0,1136	0,3205
11	0,2419	0.1341	0.1775	0,1041	0,3166
12	0,2285	0.179	0.1921	0,1051	0,4385
13	0,2258	0.2056	0.191	0,1062	0,3276
14	0,2140	0.1372	0.3108	0,1024	0,3441
15	0,2958	0.1329	0.2039	0,1348	0,3045
16	0,2263	0.13	0.1915	0,1143	0,3013
17	0,2063	0.1483	0.1913	0,9503	1,1535
18	0,2012	0.3241	0.1795	0,1109	0,3091
19	0,3748	0.1994	0.1765	0,1663	0,3158
20	0,3726	0.1374	0.1747	0,1208	0,3089
21	0,2419	0.1337	0.2255	0,1220	0,3287
22	0,2285	0.1354	0.2835	0,1248	0,3113
23	0,2258	0.1321	0.1833	0,1328	0,3126
24	0,214	0.1321	0.1783	0,1271	0,3100

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

25	0,2958	0.1321	0.1906	0,1009	0,3740
26	0,2263	0.1322	0.1823	0,1005	0,3193
27	0,2063	0.1681	0.1812	0,1007	0,3042
28	0,2012	0.2563	0.1795	0,1754	0,3345
29	0,3748	0.1485	0.188	0,1362	0,3003
30	0,3726	0.1367	0.2372	0,1206	0,3354
Rata-rata	0,347393	0.21436	0.27384	0,65302	0.627867

Tabel 6.3 Waktu Yang Dibutuhkan Untuk Penguraian Token (ms)

Pengujian	PASETO			JWT	
	PASETO v2 Custom	PASETO v2,local	PASETO v2,public	HS256	RSA256
1	0,3497	0,1548	0,2785	2,2377	12,4124
2	0,2512	0,0834	0,2161	0,1323	0,1763
3	0,2432	0,0842	0,2159	0,1817	0,1584
4	0,2866	0,1065	0,2319	0,1180	0,1609
5	0,2596	0,0993	0,2330	0,1076	0,2187
6	0,3457	0,0900	0,2256	0,1122	0,1743
7	0,2382	0,0890	0,2266	0,1196	0,1489
8	0,2402	0,0858	0,2176	0,1150	0,1484
9	0,2348	0,0846	0,2156	0,1134	0,1643
10	0,2755	0,0832	0,2139	0,1015	0,1611
11	0,2720	0,1002	0,2139	0,0954	0,1456
12	0,2320	0,1811	0,2325	0,0962	0,1744
13	0,2683	0,0964	0,2247	0,0942	0,1573
14	0,3354	0,0845	0,2452	0,2110	0,1558
15	0,2515	0,0829	0,2282	0,1129	0,1480
16	0,2378	0,0820	0,2258	0,1070	0,1777
17	0,2313	0,0947	0,2219	0,1096	0,1484
18	0,5122	0,0951	0,2168	0,1111	0,1619
19	0,4661	0,1478	0,2149	0,1216	0,1452
20	0,3563	0,1073	0,2166	0,1157	0,2695
21	0,2720	0,0868	0,2918	0,1075	0,1393
22	0,2320	0,0851	0,2616	0,1044	0,1287
23	0,2683	0,0844	0,2172	0,1242	0,1484
24	0,3354	0,0843	0,2308	0,1005	0,1395
25	0,2515	0,0840	0,2193	0,0941	0,1622
26	0,2378	0,1974	0,2178	0,0933	0,1386
27	0,2313	0,0929	0,2170	0,0936	0,1339
28	0,5122	0,1080	0,2158	0,1095	0,1438

29	0,4661	0,0884	0,3308	0,1120	0,1718
30	0,3563	0,0853	0,2309	0,1076	0,1468
Rata-rata	0,301683	0,10098	0,568683	0,185347	0,568683