Challenge Deterministik Berbasis Korpus Publik untuk Mitigasi Replay pada API Stateless

Atharizza Muhammad Athaya - 18223079
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: atharizza535@gmail.com , 18223079@std.stei.itb.ac.id

Abstrak*—*Replay attack* tetap menjadi permasalahan pada autentikasi API karena *request* yang valid sebelumnya dapat ditangkap dan dikirim ulang. Makalah ini mendesain dan mengevaluasi penerapan mekanisme *challenge* deterministik berbasis korpus publik untuk pengikatan konteks pada arsitektur API *stateless*. *Challenge* diturunkan dari metode HTTP, *path*, *query*, *body hash*, identitas klien, versi korpus, dan *time bucket*. challengeId dan challengeDigest disertakan pada *canonical request* yang ditandatangani menggunakan Ed25519. Adapun prototipe yang diajukan diimplementasikan menggunakan Node.js dan Hono.js dan dievaluasi pada mode *unsigned*, *signature-only*, dan *corpus-challenge*. Hasil menunjukkan bahwa *request* yang valid ditandatangani diterima verifikator, sementara *request* dengan *query* yang diubah, serangan antar *endpoint* yang berbeda, dan *timestamp* yang sudah kedaluwarsa ditolak. Mode *corpus-challenge* mengekspos perbedaan konteks secara eksplisit melalui verifikasi *challenge*, sementara *signature-only* mendeteksinya pada tahapan verifikasi tanda tangan. Dari 250 *request* yang diuji secara sekuensial pada setiap mode, mode *corpus-challenge* menambah latensi rata-rata sebesar 0,165 ms jika dibandingkan dengan mode verifikasi *signature-only*. Walaupun begitu, kedua mode *stateless* tetap menerima *replay* identik di dalam *time bucket* yang sama selama masih valid karena verifikator tidak menyimpan *replay memory*. Karenanya, mekanisme yang diajukan diposisikan sebagai *layer* pengikat konteks *stateless* yang secara eksplisit dapat mereproduksi *challenge* yang dibuat, daripada diangkat sebagai pengganti dari penyimpanan *nonce*, *replay caches*, atau mekanisme *idempotency* pada *endpoint* kritis.

Kata Kunci*—*replay attack, API stateless, korpus publik, challenge deterministik , Ed25519, tanda tangan digital.

I. Pendahuluan

Replay attack adalah permasalahan yang sudah ada sedari lama dalam protokol autentikasi. Pada serangan ini, penyerang akan menangkap pesan atau *request* yang valid sebelumnya dan mengirim ulang pesan tersebut untuk mendapatkan manfaat yang sama, atau melakukan impersonasi terhadap pengirim awal yang sah. Pada sistem API, permasalahan ini akan menjadi signifikan ketika sebuah *request* memicu permintaan yang bersifat sensitif, seperti perubahan atau modifikasi data, pembuatan token, konfirmasi transaksi, atau akses terhadap sumber daya yang dilindungi. Studi protokol autentikasi yang telah dilakukan oleh Needham dan Schroeder, atau Denning dan Sacco menunjukkan bahwa penggunaan ulang pesan telah menjadi perhatian utama dalam autentikasi jaringan dalam beberapa dekade terakhir [1], [2]. National Institute of Standards and Technology (NIST) juga menempatkan keamanan terhadap serangan ini menjadi hal yang penting pada mekanisme autentikasi yang diterapkan, terutama ketika penyerang dapat menangkap dan menggunakan kembali pesan yang valid [3].

Permasalahan ini akan menjadi lebih rumit pada arsitektur API *stateless*. Secara konsep, HTTP didefinisikan sebagai protokol *request/response stateless* yang berjalan di atas layer protokol aplikasi, menunjukkan bahwa setiap *request* yang dikirim akan diinterpretasikan secara independen selama aplikasi tidak menambahkan *layer* manajemen *state*-nya sendiri [4]. Pencegahan serangan *replay* yang kuat biasanya bergantung pada penyimpanan *request* valid sebelumnya, seperti penggunaan *nonce*, *idempotency key*, sidik jari (*fingerprint*), atau *replay memory*.

Makalah ini mengusulkan pendekatan mitigasi serangan *replay* pada API *stateless* dengan mekanisme *challenge* deterministik yang menggunakan korpus publik sebagai basisnya. Posisi korpus publik adalah sebagai sumber *challenge* publik yang dapat dibuat ulang, bukan sebagai rahasia, pengganti kunci kriptografi, atau entropi. Pada setiap *request* yang dibuat, klien dan server akan menurunkan *challenge* yang sama sesuai dengan konteks *request* yang dibuat, identitas klien, versi korpus publik yang digunakan, dan *bucket* waktu. Metadata *challenge* yang dibuat akan disertakan dalam *canonical request*, mencakup challengeId dan challengeDigest. Klien kemudian menandatangani *canonical request* menggunakan Ed25519. Hal ini memungkinkan server untuk memverifikasi *signature* dari *request* yang dibuat beserta dengan ikatannya terhadap *challenge* yang diturunkan sebelumnya [5], [6].

Namun, sistem yang diusulkan tidak mengklaim dapat menangani *replay* identik dalam rentang *bucket* yang sama tanpa *state* tambahan.

Adapun kontribusi makalah adalah sebagai berikut: 1) Makalah ini mendesain mekanisme *challenge* deterministik berbasis korpus publik untuk verifikasi *request* API *stateless*; 2) Makalah ini mengintegrasikan *challenge* terhadap *canonical request* yang sudah ditandatangani dengan Ed25519; 3) Makalah ini mengimplementasikan prototipe API dan lab berbasis web untuk mengobservasi alur dan perilaku klien, server, dan penyerang; dan 4) Makalah ini mengevaluasi desain yang diajukan melalui *request* yang valid, *request* yang diubah/dirusak, *timestamp* yang tidak berlaku, *replay* berbasis

cross-endpoint, pengikatan *body* protokol POST, dan serangan *replay* pada rentang *bucket* yang sama.

II. Dasar Teori

A. Replay Resistance, Freshness, Timestamp, dan Nonce

Replay resistance berkaitan dengan kemampuan protokol untuk menolak penggunaan ulang pesan yang sebelumnya valid. Dalam literatur autentikasi, konsep *freshness* menjelaskan bahwa pesan yang valid secara kriptografis masih bersifat rentan ketika dapat digunakan kembali di luar konteks atau waktu yang dimaksudkan [1], [2].

B. Autentikasi Challenge-Response

Autentikasi *challenge-response* adalah autentikasi berbasis pola pertanyaan dan jawaban. Verifikator memberikan atau menentukan *challenge*, kemudian pihak yang diautentikasi harus menghasilkan *response* yang valid dan terikat pada *challenge* tersebut. Pada berbagai protokol autentikasi, *challenge* yang dibuat bersifat acak dan/atau disimpan secara sementara oleh verifikator. Hal ini akan mencegah penggunaan ulang, disebabkan oleh *response* yang hanya valid pada *challenge* tertentu saja.

C. Tanda Tangan Digital Ed25519

Makalah ini menggunakan Ed25519 sebagai mekanisme tanda tangan digital untuk autentikasi *request* API. Ed25519 adalah salah satu contoh dari Edwards-Curve Digital Signature Algorithm pada RFC 8032 [6]. Pada skema tanda tangan digital, klien akan menandatangani pesan dengan *private key* dan server memverifikasi tanda tangan dengan *public key*.

D. Penandatanganan Request HTTP dan Canonicalization

Penandatanganan *request* HTTP akan mengikat komponen-komponennya kepada tanda tangan terkait, mencakup metode HTTP, *path*, *query*, *header*, *timestamp*, dan *body digest*. RFC 9421 mendefinisikan sebuah *framework* untuk HTTP *Message Signatures* dan memfokuskan perlunya memilih dan merepresentasikan komponen HTTP secara konsisten sebelum dilakukannya penandatanganan dan verifikasi [5]. Hal ini penting karena tanda tangan hanya melindungi komponen yang memang sudah ditandatangani.

Oleh karena itu, *canonicalization* adalah bagian dari *security surface*. Ketika klien dan server merepresentasikan *path*, *query*, *header*, atau *body* secara berbeda, maka *request* valid dapat ditolak.

E. Fungsi Hash untuk Derivasi Deterministik

Dalam makalah ini, fungsi *hash* digunakan untuk membuat ikatan deterministik. Algoritma SHA-256 digunakan untuk menghitung *hash* dari *request body*, menurunkan *seed* untuk *challenge* terkait, membuat *challenge digest*, dan mengkonstruksi *identifier* untuk *challenge* tersebut. SHA-256 diidentifikasikan sebagai bagian dari fungsi *hash* SHA-2, sebagaimana dispesifikasikan oleh Secure Hash Standard [7].

F. HTTP Stateless, Idempotency, dan Replay Memory

HTTP didefinisikan sebagai protokol *request/response stateless* yang berjalan di atas *layer* aplikasi [4]. Hal ini berarti setiap *request* akan diinterpretasikan secara independen selama tidak ada penambahan mekanisme manajemen *state* pada aplikasi terkait. Sifat *stateless* ini berguna untuk skalabilitas dan *deployment*, namun juga berarti verifikator tidak dapat mengetahui secara langsung apakah sebuah *request* sudah diproses.

Pertahanan *replay* yang bersifat *stateful* menyediakan deteksi duplikasi yang lebih kuat. Dalam hal ini, penyimpanan *nonce*, *replay cache*, *fingerprint*, atau mekanisme *idempotency* memungkinkan server untuk menolak *request* berulang. Draf *header* HTTP Idempotency-Key mendeskripsikan mekanisme klien dalam menyisipkan kunci unik kepada *request* supaya server dapat menemukan pengulangan submisi [8]. Mekanisme ini akan berguna pada operasi non-*idempotent*, namun membutuhkan *state* dari sisi server, kebijakan ekspirasi, dan sinkronisasi kuat pada *deployment* terdistribusi.

G. Korpus Publik sebagai Sumber Challenge

Pada desain yang diajukan makalah ini, korpus publik berfungsi sebagai sumber *challenge* publik. Klien dan server menggunakan konteks *request* yang sama dan *bucket* waktu untuk memilih fragmen korpus yang sama. Bagian yang terpilih akan digunakan untuk menurunkan metadata *challenge* seperti challengeDigest dan challengeId. Adapun penyerang juga dapat mengetahui korpus dan algoritma derivasi *challenge*, karena korpus tidak digunakan sebagai rahasia. Peran ini berbeda dari penggunaan korpus sebagai objek data yang hanya dilayani API, dimana korpus ikut menjadi bagian dari alur protokol sebagai sumber *challenge*.

H. Gap Penelitian

Gap yang diangkat pada makalah ini adalah bagaimana pengikatan *request* API dapat diperkaya menggunakan *challenge* deterministik dari korpus publik tanpa mengubah verifikator menjadi sistem yang bergantung pada replay memory. Desain yang diajukan mengeksplorasi apakah korpus publik dapat berfungsi sebagai sumber *challenge* yang dapat direproduksi untuk mitigasi serangan *replay* di konteks dan waktu yang berbeda.

III. Definisi Masalah dan Tujuan Desain

A. Definisi Masalah

Adapun permasalahan yang diangkat pada makalah ini adalah sebagai berikut:

Bagaimana ikatan sebuah request pada API stateless dapat diperkuat menggunakan challenge deterministik dari korpus publik sembari menjaga reprodusibilitas tanpa menerapkan replay memory untuk setiap request?

B. Tujuan Desain

Adapun desain yang diajukan makalah ini memiliki empat tujuan utama, mencakup:

1) Verifikasi *stateless*: Server dapat memverifikasi *request* tanpa menyimpan *request*, *nonce*, atau *challenge* yang telah diterima sebelumnya.
2) *Context-based binding*: Adapun tanda tangan digital yang dibuat harus mengikat metode HTTP, *path*, *query* kanonik, *body*, identitas klien, *timestamp*, dan *bucket* waktu. Setiap perubahan konteks *request* menghasilkan *request* atau *challenge* yang berbeda.

3) *Corpus-challenge binding*: *Request* yang telah ditandatangani harus mencakup metadata *challenge* yang diturunkan dari korpus publik. Korpus berfungsi sebagai penyedia *challenge* yang dapat direproduksi dan bukan sebagai penyedia rahasia autentikasi.
4) Reprodusibilitas deterministik: Server dan klien harus dapat menurunkan entri korpus, fragmen, *challenge digest*, dan *identifier* yang konsisten, jika diberikan konteks *request*, versi korpus, dan *bucket* identik.

C. Model Ancaman dan Asumsi Kepercayaan

Penyerang diasumsikan dapat mengamati dan meniru *request* API yang telah valid sebelumnya. Penyerang dapat mengirim kembali *request* tanpa dilakukannya perubahan, atau melakukan modifikasi metode, *path*, *query*, *body*, identitas klien, *timestamp*, *challenge identifier*, atau *challenge digest*. Penyerang juga diasumsikan memiliki akses terhadap korpus publik, versi korpus, mekanisme kanonik, algoritma derivasi *challenge*, versi protokol, dan *time bucket* yang diterima.

Penyerang diasumsikan untuk tidak memiliki *private key* Ed25519 milik klien. Akibatnya, penyerang tidak dapat membuat tanda tangan digital yang valid untuk *canonical request* yang dibuat di bawah asumsi standar keamanan Ed25519 [6]. Jika *private key* sudah berpindah tangan, maka penyerang dapat membuat *request* bertanda tangan yang valid, menyebabkan mekanisme yang diajukan tidak dapat membedakannya dengan klien yang sah.

Adapun *trust model* direpresentasikan pada tabel ini:

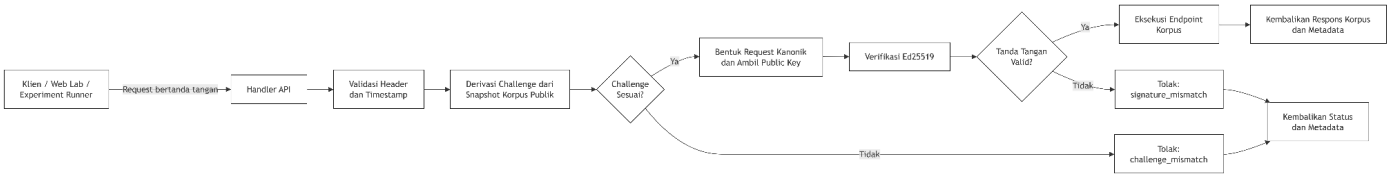
TABEL I. TRUST MODEL

Komponen	Asumsi
Private key klien	Dipegang secara aman oleh klien sah
Public key registry server	Sesuai dan dapat dipercaya
Korpus publik	Publik, memiliki versi, dan menggunakan *snapshot* konsisten antara klien dan server
Algoritma *challenge*	Publik dan deterministik
Aturan kanonik	Identik antara penandatangan dan verifikator
Penyerang jaringan	Dapat mengamati, meniru, mengubah, dan melakukan serangan *replay* request
Replay memory	Tidak tersedia pada mode verifikasi *stateless*; dapat ditambahkan sebagai kontrol terpisah

IV. METODE YANG DIAJUKAN

A. Arsitektur dan Model Evaluasi

Sistem yang diajukan mencakup klien, verifikator API, sebuah *public key registry*, *snapshot* korpus publik yang memiliki versi, dan *endpoint* API korpus. Klien membuat *request* API, menurunkan *challenge* terkait dari korpus, membuat *canonical request*, dan menandatanganinya menggunakan *private key* Ed25519. Server secara independen merekonstruksi *challenge* dan *canonical request* sebelum memverifikasi tanda tangan dengan *public key* klien yang terdaftar. Adapun arsitektur sistem yang diajukan secara garis besar ditunjukkan pada Gambar 1.



Gambar 1. Alur verifikasi sistem challenge deterministik korpus publik.

Adapun prototipe mendukung tiga mode operasi berikut:

TABEL II. MODE OPERASI

Mode	Deskripsi
UNSIGNED	*Request* diproses tanpa verifikasi kriptografik
SIGNATURE_ONLY	*Request* ditandatangani dengan Ed25519 dan divalidasi dengan *timestamp*
CORPUS_CHALLENGE	*Request* ditandatangani dengan Ed25519 dan terikat dengan *challenge* korpus deterministik

B. Model Baseline Signature-Only

Model *signature-only* sebagai *baseline* menggunakan *canonical request* yang mengandung metode HTTP, *path*, *query*, *body hash*, identitas klien, dan *timestamp*. Klien menandatangani representasi ini menggunakan *private key* Ed25519 miliknya, sementara server memverifikasinya menggunakan *public key* yang sesuai.

Misalkan M sebagai metode HTTP, P sebagai *path request*, Q_c sebagai *canonical query*, B sebagai *body*, CID sebagai identitas klien, dan T sebagai *timestamp*, maka *body hash* turunan *request* didefinisikan sebagai berikut:

$$BH = SHA256(B)$$

Canonical request dalam mode *signature-only* kemudian direpresentasikan sebagai berikut:

$$CR_{sig} = M \,||\, P \,||\, Q_c \,||\, BH \,||\, CID \,||\, T$$

Klien kemudian menandatangani *request*:

$$\sigma = Ed25519.Sign(SK_{klien}, CR_{sig})$$

Server kemudian memverifikasi tanda tangan tersebut:

$$Ed25519.Verify(PK_{klien}, CR_{sig}, \sigma)$$

C. Pembentukan Canonical Request

Canonicalization memastikan bahwa server dan klien mengkonstruksi representasi yang sama dan tertandatangani dari *request* HTTP yang sama. Prototipe ini menormalisasi metode HTTP menjadi *uppercase*, menggunakan / ketika tidak ada *path* yang diberikan, mengurutkan parameter *query* dari *key* dan *value*, menerapkan *percent-encoding*, dan melakukan *hashing* pada *body request* menggunakan SHA-256.

Adapun *canonical query* didefinisikan sebagai berikut:

$$Q_c = Encode(Sort(Q))$$

Dalam mode CORPUS_CHALLENGE, *canonical request* mencakup setidaknya komponen-komponen berikut:

1) HTTP_METHOD
2) PATH
3) CANONICAL_QUERY
4) BODY_HASH
5) client-id:CLIENT_ID
6) timestamp:TIMESTAMP
7) challenge-id:CHALLENGE_ID
8) challenge-digest:CHALLENGE_DIGEST

D. Derivasi Challenge Berbasis Korpus Publik

Challenge deterministik diturunkan dari konteks *request*, versi korpus, versi protokol, dan *time bucket*. Misalkan Δ sebagai durasi *bucket* dalam detik, maka *time bucket* dikalkulasikan sebagai berikut:

$$TB = floor(\frac{T}{\Delta})$$

Implementasi berikut menggunakan durasi bawaan sebesar 30 detik. Kesegaran *timestamp* akan divalidasi secara terpisah menggunakan perbedaan maksimum 60 detik.

Dengan CV sebagai versi korpus, maka konteks *endpoint* dikonstruksi dari:

$$EC = M \,||\, P \,||\, Q_c \,||\, BH \,||\, CID \,||\, TB \,||\, CV$$

Kemudian, sebuah *seed* dikalkulasi dengan versi PV:

$$S = SHA256(PV \,||\, "seed" \,||\, EC)$$

Dengan N adalah jumlah entri korpus, paruh pertama *seed* akan memilih satu entri korpus:

$$I = Integer(S_{0:32}) \bmod N$$

Kemudian, paruh sisanya menentukan *offset* fragmen:

$$O = Integer(S_{32:64}) \bmod |Corpus_I|$$

Setelahnya, fragmen F dengan panjang tetap akan dibaca dari entri korpus terpilih. Implementasi dilakukan menggunakan *circular reading* ketika fragmen mencapai akhir dari teks korpus. Panjang fragmen bawaan adalah 96 karakter.

Challenge digest kemudian dikalkulasikan sebagai:

$$CD = SHA256(PV \,||\, CV \,||\, CorpusId_I \,||\, O \,||\, F \,||\, TB)$$

Identifier challenge yang mengikat *digest* kembali ke konteks *endpoint* dihitung dengan:

$$CI = SHA256(PV \,||\, "challenge - id" \,||\, CD \,||\, EC)$$

Metadata yang dihasilkan mencakup komponen berikut ini:

1) protocolVersion;
2) corpusVersion;
3) challengeId;
4) challengeDigest;
5) corpusId;
6) fragmentOffset;
7) fragmentPreview;
8) timeBucket.

E. Penandatanganan Request

Setelah dilakukannya derivasi *challenge*, klien menarik challengeId dan challengeDigest pada *canonical request*. Klien kemudian menandatangani representasi penuh.

$$CR_{challenge} = CR_{sig} \,||\, CI \,||\, CD$$

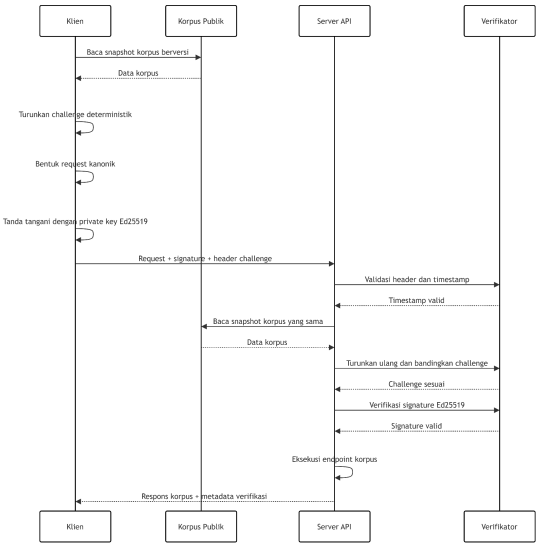
$$\sigma = Ed25519.Sign(SK_{klien}, CR_{challenge})$$

Adapun *request* yang ditandatangani memiliki *header* ini:

1. X-Client-Id
2. X-Timestamp
3. X-Signature
4. X-Challenge-Id
5. X-Challenge-Digest

Fragmen yang terpilih tidak dikirimkan sebagai bahan autentikasi. Server dan klien akan menurunkannya secara independen dari *snapshot* korpus dan konteks *request* yang sama. *Header challenge* yang terlihat mengizinkan server untuk membandingkan *challenge* yang diberikan dengan nilai *challenge* yang diturunkan secara independen.

Alur penandatanganan dan verifikasi request yang valid ditunjukkan pada Gambar 2 yang merepresentasikan interaksi antara klien, server API, korpus publik, dan verifikator terhadap *request* yang valid berdasarkan ikatan *challenge*.



Gambar 2. Alur request valid yang diikat pada challenge korpus publik

F. Status Verifikasi

Adapun implementasi akan menghasilkan alasan kegagalan secara eksplisit sebagai berikut:

TABEL III. STATUS KEGAGALAN

Status	**Deskripsi**
missing_client_id	Identitas klien kosong
missing_timestamp	*Timestamp* kosong
invalid_timestamp	*Timestamp* bukan *integer* valid
timestamp_expired	*Request* berada di luar toleransi kesegaran
unknown_client	Tidak terdapat *public key* untuk klien
missing_signature	Tanda tangan digital kosong
missing_challenge_id	challengeId kosong
missing_challenge_digest	challengeDigest kosong
challenge_mismatch	Perbedaan antara metadata *challenge* dengan hasil rekalkulasi independen server
signature_mismatch	Verifikasi Ed25519 gagal.

G. Batasan Replay dalam Time Bucket yang Sama

Metode yang diajukan mengikat *request* kepada konteks dan waktu, namun tidak menerapkan *replay memory*. Ketika penyerang melakukan serangan *replay* identik di *time bucket* yang valid, maka server juga menerima metode, *path*, *query*, *body*, *timestamp*, metadata *challenge*, dan tanda tangan digital yang sama. Hal ini kemudian akan mengkonstruksi *canonical request* yang sama.

Verifikator *stateless* tidak memiliki tambahan nilai yang dapat membedakan *request* yang asli dengan yang dikirimkan ulang. Penolakan duplikat membutuhkan *replay memory*, seperti *nonce*, *idempotency*, atau validasi bisnis.

Batasan ini mendefinisikan ruang lingkup kontribusi: mitigasi pengulangan di berbagai konteks *request* dan *time bucket* melalui pengikatan *challenge* deterministik, bukan deteksi duplikat absolut.

V. IMPLEMENTASI PROTOTIPE

A. Struktur Perangkat Lunak

Metode yang diajukan diimplementasikan sebagai prototipe API eksperimental menggunakan *runtime* Node.js dan *framework* Hono.js. Implementasi memisahkan penerapan protokol dalam beberapa modul terpisah: 1) Modul *canonicalization* menormalisasikan parameter *query* dan mengkonstruksi *canonical request*; 2) modul *challenge* menghitung *time bucket*, memilih entri korpus dan *offset* fragmen, dan melakukan derivasi challengeId dan challengeDigest; 3) modul keamanan yang melakukan penandatanganan dan verifikasi Ed25519; dan 4) modul aplikasi yang menyediakan *route* API dan menerapkan *middleware* verifikasi untuk *endpoint* korpus yang dilindungi.

Prototipe dirancang menggunakan *snapshot* korpus lokal yang telah diberikan versi, diidentifikasikan sebagai public-corpus-v1. Konfigurasi bawaan prototipe menggunakan *time bucket* berdurasi 30 detik, toleransi kesegaran *request* selama 60 detik, dan fragmen korpus dengan panjang 96 karakter. Nilai-nilai tersebut dapat dikonfigurasikan kembali tanpa mengubah algoritma derivasi *challenge*.

Prototipe menyediakan *key pair* Ed25519 di dalam memori dan mendaftarkan satu klien sebagai demo-client. Kunci berikut dirancang untuk hanya digunakan di level eksperimental. Penerapan *key pair* yang disediakan tidak mengikuti standar manajemen *key* yang mengharuskan *private key* terjaga di sisi klien dan *public key registry* yang menggunakan mekanisme terpercaya.

B. API dan Web Lab

Adapun API yang disediakan akan melindungi tiga operasi korpus, sebagaimana ditunjukkan pada tabel berikut:

TABEL IV. API ENDPOINT

Endpoint	**Fungsi**
GET /corpus	Mengembalikan atau mencari koleksi korpus
GET /corpus/:id	Mengembalikan hanya satu entri korpus sesuai ID
POST /corpus/query	Mencari korpus menggunakan *signed request*

Middleware verifikasi diimplementasikan pada *endpoint* /corpus dan turunannya. Perilaku *endpoint* ini akan bergantung kepada mode keamanan yang dikonfigurasikan sebelumnya, yaitu mode UNSIGNED, SIGNATURE_ONLY, atau CORPUS_CHALLENGE. Respons terhadap *request* yang diterima memuat mode aktif, status verifikasi, *hash canonical request*, *time bucket*, dan metadata *challenge*. *Request* yang ditolak memiliki alasan kegagalan yang eksplisit disebutkan, seperti timestamp_expired, challenge_mismatch, atau signature_mismatch.

Endpoint /challenge mengekspos metadata *challenge* deterministik untuk pengamatan. Bagian ini tidak dibutuhkan oleh protokol utama, karena klien dan server dapat melakukan derivasi secara independen. Adapun *endpoint* /health memberikan status mode keamanan yang sedang aktif dan digunakan oleh kontainer *health check*.

Prototipe yang diajukan juga menyediakan lab berbasis *browser* yang dapat dibuka dari *endpoint* /lab. *Endpoint* ini merepresentasikan panel di masing-masing pihak, baik dari klien, server, dan penyerang. Panel klien dapat menciptakan dan mengirimkan *request* yang telah ditandatangani, sementara panel penyerang dapat mengirimkan ulang *request* yang sama, mengubah isi *query*, mengganti *timestamp* dengan nilai yang sudah kedaluwarsa, atau memindahkannya ke *endpoint* yang lain. Panel server menunjukkan apakah *request* yang dilakukan diterima atau ditolak, mencakup alasan verifikasi yang eksplisit.

Sebagai kebutuhan demonstrasi, *endpoint* /lab/sign akan melakukan penandatanganan *request* menggunakan *key pair* demonstrasi yang disediakan server. Karenanya, lab yang disediakan hanya mensimulasikan proses penandatanganan logis dan tidak mengimplementasikan klien yang menyimpan *private key*-nya secara independen.

C. Pengujian dan Reprodusibilitas

Prototipe menyediakan pengujian unit dan pengujian level API yang terotomasi menggunakan Vitest, dengan V8 untuk pengukuran *coverage*. Disediakan juga program eksperimental yang berjalan untuk mengevaluasi skenario dalam mode SIGNATURE_ONLY dan CORPUS_CHALLENGE. Keluaran eksperimen yang dilakukan akan disimpan dalam format JSON dan CSV untuk analisis lanjut.

Prototipe dibungkus dengan kontainer Docker dan Docker Compose. Kontainer mengekspos API yang diimplementasikan pada port 8787 dan menggunakan *endpoint* /health sebagai *health check* sistem. Diimplementasikan juga alur kerja GitHub Actions yang menginstal dependensi, menjalankan tes yang terotomasi, memberikan laporan *coverage*, dan menjalankan program eksperimental. Komponen-komponen ini memungkinkan prototipe dan segala eksperimen yang dilakukan untuk direproduksi ulang pada lingkungan yang konsisten.

VI. Rancangan Evaluasi

A. Tujuan dan Variabel Evaluasi

Evaluasi yang dilakukan mencakup tiga aspek pada metode yang diajukan, yaitu kesesuaian fungsional, pengikatan *request* berdasarkan konteks, dan batasan *replay* terhadap verifikasi *stateless*. Eksperimen yang dilakukan membandingkan mode SIGNATURE_ONLY dan CORPUS_CHALLENGE. Adapun mode UNSIGNED tidak akan dievaluasi baik secara manual ataupun disertakan pada program eksperimental terotomasi.

Variabel independen yang dipertimbangkan pada evaluasi ini adalah mode keamanan, skenario *request*, dan durasi *time bucket* yang telah dikonfigurasi lebih lanjut. Keluaran yang diamati mencakup status HTTP, alasan verifikasi, latensi pemrosesan *request*, jumlah penolakan *request*, rata-rata latensi, dan latensi di persentil ke-95.

B. Skenario Pengujian

Adapun program eksperimental terotomasi menjalankan skenario-skenario berikut ini:

TABEL V. Skenario Evaluasi

Skenario	**Prosedur**
Signed request valid	Mengirimkan *signed request* yang tidak diubah
Query diubah	Perubahan *query* setelah ditandatangani
Timestamp kedaluwarsa	Menandatangani dengan waktu di masa lampau
Antar *endpoint*	Menggunakan *signed request* yang sama pada *endpoint* yang lainnya
Request pertama pada rentang waktu yang sama	Mengirimkan *request* yang valid dan tidak dimodifikasi
Serangan *replay* pada rentang waktu yang sama	Mengirimkan kembali *request* yang identik dari sebelumnya
Request volume tinggi	Mengirimkan 250 *request* unik yang valid
Konfigurasi *bucket*	Mengirimkan *request* valid dengan durasi *bucket* 10, 30, dan 60 detik

Timestamp diatur 120 detik di belakang waktu server, menyebabkan *request* yang dihasilkan melewati toleransi kesegaran 60 detik.

C. Pengukuran Runtime dan Batasan Interpretasi

Skenario fungsional dieksekusi menggunakan antarmuka *request in-process* milik Hono.js. Penandatanganan yang dilakukan dari sisi klien diselesaikan sebelum pengukuran latensi dilakukan. Latensi elapsedMs yang diukur merepresentasikan *request handling* dari server, mencakup verifikasi dan eksekusi *endpoint*, namun tidak mencakup penandatanganan klien, transmisi jaringan, pemrosesan TLS, penjadwalan kontainer, dan akses *storage* eksternal.

Pada skenario *request* dengan volume yang tinggi, program mengirimkan 250 *request* dengan nilai *query* yang unik. Kemudian, program akan merekam jumlah *request* yang ditolak, rata-rata latensi pemrosesan server, dan latensi pada persentil ke-95 pada setiap mode keamanan. Hasil pengukuran tersebut akan memberikan perbandingan antara kedua mode SIGNATURE_ONLY dan CORPUS_CHALLENGE, namun keluaran tidak ditujukan untuk diinterpretasikan sebagai *throughput benchmark* sistem.

Skenario *bucket* dengan durasi yang berbeda-beda, mulai dari 10, 30, dan 60 detik akan mengeksekusi masing-masing satu *request* yang valid.

VII. Hasil Dan Analisis

A. Kesesuaian Fungsional dan Pengikatan Konteks

Rangkaian otomasi pengujian yang diselesaikan menunjukkan 16 pengujian lolos di dua berkas pengujian. Pengujian yang dilakukan mengkonfirmasi derivasi *challenge* deterministik untuk masukan identik, perubahan nilai *challenge* pada berbagai konteks *request* dan *time bucket*, verifikasi Ed25519 yang valid, penolakan *request* yang telah diubah, ekspirasi *timestamp*, perilaku API yang dilindungi, dan limitasi serangan *replay* pada rentang waktu yang sama. Adapun hasil eksperimen dirangkum pada Tabel VI.

TABEL VI. Keluaran Fungsional dan Replay

Skenario	**SIGNATURE_ONLY**	**CORPUS_CHALLENGE**
Signed request valid	200 accepted	200 accepted
Query diubah	401 signature_mismatch	401 challenge_mismatch

Timestamp kedaluwarsa	401 timestamp_expired	401 timestamp_expired
Antar *endpoint*	401 signature_mismatch	401 challenge_mismatch
Request pertama pada rentang waktu yang sama	200 accepted	200 accepted
Serangan *replay* pada rentang waktu yang sama	200 accepted	200 accepted

Kedua mode keamanan menerima *request* yang telah ditandatangani dan menolak *query* yang telah dimodifikasi atau skenario serangan antar *endpoint*. Perbedaan ditunjukkan pada tahapan verifikasi yang menyebabkan penolakan. Pada mode SIGNATURE_ONLY, *request* yang telah dimodifikasi membuat representasi kanonik yang berbeda dan verifikasi Ed25519 yang gagal dengan status signature_mismatch. Pada mode CORPUS_CHALLENGE, perubahan konteks *request* menghasilkan *challenge* deterministik yang berbeda dan ditolak lebih awal dengan status challenge_mismatch.

Request dengan *timestamp* 120 detik di belakang waktu server ditolak dengan status timestamp_expired di kedua mode. Hal ini menunjukkan konsistensi dengan konfigurasi toleransi kesegaran *request* selama 60 detik.

B. Batas Replay Stateless

Request pertama dan *request replay* yang identik diterima di kedua mode. Kedua *request* memiliki metode, *path*, *query*, *body*, *timestamp*, metadata *challenge*, dan tanda tangan yang identik pada mode yang menggunakannya. Karena verifikator tidak menyimpan *replay memory*, verifikator akan merekonstruksi *canonical request* yang sama untuk kedua *request* dan hasil verifikasi yang sama.

C. Pengamatan Runtime

Skenario volume tinggi menjalankan 250 *request* yang unik di setiap mode keamanan. Tidak ada *request* yang ditolak. Adapun pengamatan latensi server dalam prosesnya dirangkum pada Tabel VII.

TABEL VII. HASIL RUNTIME VOLUME TINGGI

	SIGNATURE_ONLY	**CORPUS_CHALLENGE**
Request	250	250
Request Ditolak	0	0
Rata-rata latensi	0,579 ms	0,744 ms
Latensi p95	1,386 ms	1,680 ms

Pada eksperimen ini, mode CORPUS_CHALLENGE menunjukkan perbedaan rata-rata latensi di angka 0,165 ms (28.6%) relatif kepada mode SIGNATURE_ONLY. Latensi p95 tercatat 0,294 ms atau sekitar 21.2% lebih lambat.

D. Validasi Konfigurasi Time Bucket

Request valid pada mode CORPUS_CHALLENGE menggunakan durasi 10, 30, dan 60 detik pada konfigurasi *bucket*. Hasil tersebut mengkonfirmasi bahwa prototipe mendukung setiap konfigurasi tanpa perubahan algoritma derivasi *challenge*.

VIII. DISKUSI

A. Interpretasi Pengikatan Challenge

Kontribusi CORPUS_CHALLENGE terlihat pada struktur verifikasi dan perannya pada pengikatan konteks. Di dalam mode SIGNATURE_ONLY, *request* yang telah dimodifikasi akan dideteksi pada verifikasi Ed25519 dan menghasilkan signature_mismatch. Mode CORPUS_CHALLENGE melakukan derivasi *challenge* yang diharapkan dari konteks yang diubah terlebih dahulu, menghasilkan status challenge_mismatch. *Challenge* yang dibuat memberikan protokol objek eksplisit dari entri korpus mana yang dipilih, *offset* fragmen, versi korpus, konteks *request*, dan *time bucket* yang dapat diamati.

Perbedaan diantara kedua mode tidak boleh dianggap sebagai penambahan kerahasiaan atau kekuatan autentikasi kriptografis. Korpus publik yang digunakan, algoritma derivasi *challenge*, challengeId, dan challengeDigest dibuat publik. Autentikasi tetap bergantung kepada *private key* Ed25519 milik klien, sementara korpus berfungsi sebagai sumber *challenge* deterministik.

B. Perbandingan dengan Mekanisme Stateful

Mekanisme *stateful* memberikan deteksi duplikasi yang lebih kuat. Salah satu contoh yang dapat diterapkan adalah mekanisme *idempotency*, yang memungkinkan klien untuk menyertakan kunci unik yang dapat disimpan atau diasosiasikan oleh server dengan *request* yang telah diproses [8]. Adapun implementasi lain seperti penyimpanan *nonce*, *replay cache*, dan validasi bisnis juga memberikan kemampuan yang serupa pada server.

Metode *challenge* yang diajukan tidak boleh menggantikan kontrol *stateful* pada *endpoint* yang memerlukan eksekusi paling banyak satu kali. Fungsi metode ini hanya memberikan pengikatan *stateless* kepada konteks *request* dan *time bucket* ketika mengekspos metadata *challenge* yang secara independen dapat direproduksi kembali.

C. Trade-off dan Asumsi Implementasi

Overhead yang diamati bersifat lokal dan tidak dapat digeneralisasi sebagai performa produksi karena pengukuran tidak mencakup transmisi jaringan, penandatanganan klien, pemrosesan TLS, penjadwalan kontainer, dan *storage* eksternal.

Operasi yang sesuai bergantung penuh pada konsistensi antara klien dan server. Kedua pihak harus menggunakan *snapshot* korpus, versi korpus, algoritma derivasi *challenge*, dan aturan *canonicalization* yang sama. Perbedaan dalam pengurutan *query*, *encoding*, normalisasi *path*, representasi *body*, atau konten korpus dapat menyebabkan *request* yang valid untuk ditolak. Dependensi ini konsisten dengan persyaratan umum tanda tangan komponen HTTP yang dapat

direpresentasikan secara konsisten oleh penandatangan dan verifikator [5].

D. Ancaman terhadap Validitas

Evaluasi yang dilakukan memiliki beberapa limitasi. Pertama, prototipe menggunakan demonstrasi korpus secara lokal, bukan sebuah *snapshot* dari penyedia korpus publik eksternal yang memiliki *provenance* dan versi yang jelas. Hasil yang diberikan juga menunjukkan pemrosesan korpus deterministik lokal, namun tidak menunjukkan perilaku sistem terhadap korpus yang lebih besar, aturan normalisasi yang berbeda, atau perubahan data eksternal yang digunakan.

Adapun limitasi kedua pada prototipe ini adalah penggunaan *key pair* yang disimpan di dalam memori. Evaluasi yang dilakukan tidak mengukur manajemen *public key registry*, rotasi kunci, penerapan sistem konkuren antar berbagai klien, atau penyimpanan *private key* yang independen di sisi klien. Lab berbasis *browser* yang disediakan juga melakukan penandatanganan lewat demonstrasi di sisi *endpoint* demonstrasi pada server dan tidak akan dinilai sebagai desain manajemen kunci di level produksi.

Limitasi ketiga mengangkat permasalahan hasil *runtime* yang hanya melakukan satu *batch* pada setiap modenya, dengan masing-masing *batch* memiliki 250 *request* unik. Eksperimen tidak mengukur *throughput* konkuren, latensi jaringan, perilaku *deployment* di sisi produksi, atau variasi statistik dari beberapa kali eksekusi.

Limitasi terakhir pada prototipe ini menunjukkan bahwa perbedaan waktu *bucket* pada eksperimen hanya menggunakan satu *request* yang valid untuk masing-masing 10, 30, dan 60 detik. Tes yang dilakukan mengkonfirmasi bahwa terdapat dukungan konfigurasi *bucket* yang berbeda, namun tidak dapat menunjukkan durasi *bucket* yang optimal, toleransi *clock skew*, probabilitas kesalahan penolakan, atau pengeksposan serangan *replay* yang efektif. Adapun modifikasi isi *body* dan serangan *replay* lintas klien juga tidak diisolasikan sebagai skenario eksperimen.

IX. Kesimpulan

Makalah ini mendesain dan mengevaluasi mekanisme *challenge* deterministik berbasis korpus publik untuk pengikatan konteks *request* pada API *stateless*. Metode yang diajukan menurunkan *challenge* eksplisit dari metode HTTP, *path*, *query*, *body hash*, identitas klien, versi korpus, dan *time bucket*. challengeId dan challengeDigest yang dihasilkan dimasukkan ke *canonical request* yang ditandatangani menggunakan Ed25519.

Evaluasi menunjukkan bahwa kedua mode menolak perubahan konteks *request* pada tahapan verifikasi yang berbeda. Metode yang diajukan mendemonstrasikan pengikatan *challenge* yang dapat diamati, tetapi tidak menunjukkan cakupan penolakan serangan yang lebih luas daripada mode SIGNATURE_ONLY.

Request yang sudah kedaluwarsa ditolak dari validasi *timestamp*, sedangkan request *replay* yang identik dalam time *bucket* yang sama tetap diterima pada kedua mode *stateless*. Hasil ini menunjukkan bahwa *challenge* deterministik yang diajukan tidak dapat membuktikan apakah *request* yang identik sudah diproses sebelumnya ketika verifikator tidak menyimpan *replay memory*. Metode yang diajukan diposisikan sebagai pengikatan konteks *request* yang berjalan secara *stateless* dan bukan sebagai pengganti mekanisme *idempotency*, penyimpanan *nonce*, *replay cache*, atau validasi bisnis pada *endpoint* penting.

Video Youtube

Video demonstrasi implementasi dan pengujian sistem dapat diakses melalui tautan berikut:

https://r.atharizza.com/yt-kripto

Repositori

Kode sumber, dokumentasi, dan hasil eksperimen dapat diakses melalui repositori berikut:

https://r.atharizza.com/gh-kripto

Ucapan Terima Kasih

Segala puji dan syukur dipanjatkan kepada Tuhan Yang Maha Kuasa karena tanpa rahmat-Nya, penulis tidak dapat menyelesaikan makalah ini. Ucapan terima kasih secara khusus disampaikan kepada Dr. Ir. Rinaldi Munir, M.T., atas bimbingan yang diberikan selama mengampu mata kuliah II4021 Kriptografi pada semester ini. Penulis juga mengucapkan terima kasih kepada orang tua yang terus memberikan dukungan selama proses perkuliahan.

Referensi

[1] R. M. Needham and M. D. Schroeder, “Using encryption for authentication in large networks of computers,” 1978.

[2] D. E. Denning and G. M. Sacco, “Timestamps in Key Distribution Protocols,” 1981.

[3] NIST SP 800-63B, Authentication and Lifecycle Management: https://pages.nist.gov/800-63-4/sp800-63b.html

[4] RFC 9110, HTTP Semantics: https://www.rfc-editor.org/rfc/rfc9110.html

[5] RFC 9421, HTTP Message Signatures: https://www.rfc-editor.org/rfc/rfc9421.html

[6] RFC 8032, EdDSA / Ed25519: https://www.rfc-editor.org/info/rfc8032/

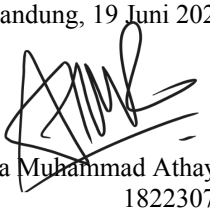
[7] NIST FIPS 180-4, Secure Hash Standard: https://csrc.nist.gov/pubs/fips/180-4/upd1/final

[8] IETF Idempotency-Key HTTP Header draft: https://datatracker.ietf.org/doc/html/draft-ietf-httpapi-idempotency-key-header

Pernyataan

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

Bandung, 19 Juni 2026



Atharizza Muhammad Athaya
18223079