Polygon: Jurnal Ilmu Komputer dan Ilmu Pengetahuan Alam
Volume. 4 Nomor. 1 Januari 2026
e-ISSN: 3046-5419; p-ISSN: 3032-6249, Hal 74-86
DOI: https://doi.org/10.62383/polygon.v4i1.931
Tersedia: https://journal.arimsi.or.id/index.php/Polygon

Pembatasan Laju Adaptif Berbasis *Verifiable Delay Function* untuk Mitigasi Penyalahgunaan API pada *Gateway Edge* Ringan

Diah Putri Kartikasari[1*], Tiara Ayu Triarta Tambak[2], Agung Nugroho[3], Ibnu Rusydi[4]
[1-4] Ilmu Komputer, Universitas Islam Negeri Sumatera Utara Medan, Indonesia
**Penulis Korespondensi: dputrikss@gmail.com*

Abstract. *API abuse on lightweight edge gateways has intensified as microservice-based services expose many REST endpoints to heterogeneous clients. Conventional per-identity rate limiting, such as static token buckets, is frequently bypassed through distributed bots and identity rotation, while legitimate burst traffic may be rejected and degrade user experience. This study proposes Adaptive Rate Limiting with Verifiable Delay Functions (ARL-VDF), which couples a lightweight risk score with selective VDF challenges to impose a tunable sequential-computation cost on suspicious clients without forcing aggressive dropping for low-risk users. The gateway continuously derives a per-identity risk score from short-window request rate, error tendency, and identity freshness, then maps the score to a target delay bounded by* τ_{min} *and* τ_{max}*. Evaluation uses a 600-second discrete-event simulation on a mixed workload consisting of normal clients, legitimate bursts, and distributed attackers. Compared with a static token bucket baseline, ARL-VDF maintains full success for legitimate traffic, reduces attacker throughput that passes the gateway, and keeps verification overhead within a fixed budget on the edge device. The results indicate that combining adaptive control with verifiable sequential cost can improve availability and fairness on resource-constrained edge gateways without resorting to aggressive dropping.*

Keywords: *Adaptive Rate Limiting; API Abuse; Edge Gateway; Risk Score; Verifiable Delay Function.*

Abstrak. Penyalahgunaan API pada *gateway edge* ringan meningkat seiring arsitektur mikroservis yang mengekspos banyak endpoint kepada klien yang beragam. Pembatasan laju statis per identitas, seperti token *bucket*, relatif mudah dilewati melalui bot terdistribusi dan rotasi identitas, sementara lonjakan trafik sah berpotensi ikut tertolak dan menurunkan kualitas layanan. Penelitian ini mengusulkan ARL-VDF, yaitu pembatasan laju adaptif yang menggabungkan skor risiko ringan dengan tantangan *Verifiable Delay Function* secara selektif untuk memberikan "harga komputasi sekuensial" pada klien berisiko tanpa memaksa penolakan masif pada pengguna normal. *Gateway* menghitung skor risiko per identitas berdasarkan laju permintaan pada jendela pendek, kecenderungan galat, dan kebaruan identitas, lalu memetakannya menjadi target penundaan yang dibatasi oleh τ_{min} dan τ_{max}.. Evaluasi dilakukan menggunakan simulasi peristiwa diskrit selama 600 detik dengan beban campuran: trafik normal, *burst* trafik sah, dan serangan terdistribusi. Dibanding *baseline* token *bucket* statis, ARL-VDF mempertahankan keberhasilan penuh untuk trafik sah, menekan throughput penyerang yang lolos, serta menjaga *overhead* verifikasi dalam anggaran tetap pada perangkat *edge*. Temuan ini menunjukkan bahwa kombinasi kontrol adaptif dan biaya komputasi sekuensial yang dapat diverifikasi efektif meningkatkan ketersediaan dan keadilan layanan pada *gateway edge* yang terbatas sumber daya.

Kata Kunci: API *Abuse*; *Edge Gateway*; Pembatasan Laju Adaptif; Skor Risiko; *Verifiable Delay Function.*

1. LATAR BELAKANG

Edge computing menempatkan pemrosesan lebih dekat ke sumber data untuk memenuhi tuntutan latensi rendah, privasi, serta efisiensi *bandwidth*, terutama pada ekosistem IoT yang heterogen (Shi et al., 2016). Dalam praktik, *gateway edge* ringan kerap berfungsi sebagai penghubung antara perangkat IoT dan layanan *cloud* sehingga memegang peran kritis terhadap keamanan dan ketersediaan layanan (Morabito et al., 2019). Seiring pergeseran arsitektur ke *microservices*, API *gateway* semakin lazim dipakai sebagai gerbang kontrol akses, pengaturan eksposur layanan, dan pengelolaan trafik, termasuk pada lingkungan *edge* yang sumber dayanya terbatas (Xu et al., 2019). Tren pemecahan layanan menjadi komponen yang lebih kecil di sisi *edge* juga memperluas permukaan serangan sekaligus meningkatkan kompleksitas

Naskah Masuk: 23 November 2025; Revisi: 21 Desember 2025; Diterima: 18 Januari 2026; Terbit: 23 Januari 2026

pengelolaan kebijakan dan dependensi antarlayanan (Hossain et al., 2023). Pedoman keamanan untuk aplikasi berbasis *microservices* menekankan perlunya perlindungan terhadap *overload* dan penyalahgunaan pada lapisan API serta penerapan kontrol berbasis kebijakan sebagai bagian dari pertahanan berlapis (Chandramouli et al., 2019).

Permasalahan utama muncul ketika API publik atau semi-publik menjadi sasaran penyalahgunaan, seperti *scraping* agresif, *credential stuffing*, eksploitasi *endpoint* yang mahal secara komputasi, serta pola *request* abnormal yang menguras sumber daya *gateway* dan *backend*. Pendekatan deteksi berbasis pola perilaku dapat membantu mengidentifikasi gejala penyalahgunaan, tetapi perlindungan ketersediaan layanan tetap memerlukan mekanisme mitigasi *real time* agar SLA tidak terdegradasi (Prinakaa et al., 2024). Selain itu, kajian kerentanan REST API menunjukkan bahwa penyalahgunaan API sering berkaitan dengan kelemahan kontrol akses, validasi *input* yang tidak memadai, dan ketiadaan proteksi trafik yang efektif pada jalur masuk layanan (Tanveer et al., 2025). Dengan demikian, kebutuhan kontrol trafik di *gateway* bukan hanya persoalan pembatasan laju, melainkan juga menjaga keadilan layanan bagi klien sah saat terjadi lonjakan beban.

Strategi yang banyak diadopsi di industri adalah *rate limiting* statis, misalnya token *bucket* per identitas. Namun pola adopsi *rate limit* pada sistem nyata tidak seragam dan konfigurasi kebijakan sering bervariasi, baik pada level platform maupun per *endpoint*, sehingga perilaku limit yang dihadapi klien cenderung heterogen (Serbout et al., 2023). Dampak kebijakan tersebut tidak selalu netral: studi menunjukkan rate limit dapat memengaruhi reliabilitas arsitektur *microservices*, misalnya melalui *retry storm*, peningkatan latensi, dan degradasi stabilitas ketika konfigurasi tidak selaras dengan dinamika beban dan ketergantungan layanan (El Malki et al., 2022). Dari sisi implementasi, pemilihan algoritma *limiter* dan parameternya melibatkan *trade-off fairness*, kemampuan menahan *burst*, serta biaya memori dan kompleksitas operasional (Bartkov & Borovikov, 2022). Bahkan untuk klien yang berperilaku baik, kebijakan statis yang kaku dapat mendorong kebutuhan mekanisme resiliensi tambahan di sisi klien, yang pada akhirnya memperbesar kompleksitas dan beban pengembangan (Sanchez et al., 2018).

Keterbatasan lain yang krusial adalah karakter serangan terdistribusi yang mudah diparalelkan. Pembatasan per identitas dapat dilewati dengan memperbanyak bot atau memutar identitas/token, sehingga *throughput* serangan yang lolos tetap tinggi, sementara trafik sah berpotensi ikut terdampak pada kondisi *burst*. Berangkat dari kesenjangan tersebut, penelitian ini bertujuan merancang dan mengevaluasi mekanisme pembatasan laju adaptif yang mampu membedakan risiko trafik secara dinamis pada *gateway edge* ringan, serta menekan

kemampuan penyerang mempertahankan laju tinggi tanpa mendorong penolakan masif terhadap pengguna sah.

2. KAJIAN TEORITIS

Rate limiting pada API *gateway* dipahami sebagai mekanisme pengendalian akses yang membatasi laju permintaan agar kapasitas layanan tidak didominasi oleh satu identitas atau satu pola trafik. Dalam praktik, variasi adopsi *rate limiting* menunjukkan bahwa kebijakan tidak hanya ditentukan oleh pilihan algoritme, melainkan juga oleh cara organisasi mengintegrasikannya pada arsitektur, pola *endpoint*, serta kebutuhan produk, sehingga konfigurasi statis sering bertahan karena mudah diterapkan meskipun tidak selalu adaptif (Serbout et al., 2023). Pada kondisi beban tinggi, kebijakan yang tidak tepat dapat memicu ketidakstabilan sistem, khususnya pada *microservices*, karena *retry*, *timeouts*, dan propagasi kegagalan antarlayanan dapat memperburuk reliabilitas keseluruhan (El Malki et al., 2022). Studi pemilihan algoritme *rate limiter* menegaskan bahwa token *bucket*, *leaky bucket*, dan *sliding window* memiliki perilaku yang berbeda dalam menghadapi *burst* dan *fairness*; konsekuensinya, keberhasilan mitigasi tidak hanya bergantung pada algoritme, tetapi juga pada kesesuaian parameter terhadap konteks operasional (Bartkov & Borovikov, 2022). Di sisi konsumen API, pendekatan resiliensi klien menekankan strategi adaptif ketika menghadapi limit, tetapi pendekatan ini tidak menghilangkan kebutuhan kontrol yang kuat pada sisi *gateway* ketika berhadapan dengan penyalahgunaan terdistribusi (Sanchez et al., 2018).

Untuk mengatasi penyalahgunaan yang bersifat masif, literatur keamanan memperkenalkan gagasan *puzzle* sebagai bentuk *cost shifting*, yaitu klien diminta membayar biaya komputasi sebelum permintaan dilayani sehingga serangan menjadi mahal untuk dipertahankan, sementara verifikasi di *server* tetap ringan (Ali et al., 2020). *Verifiable Delay Function* (VDF) memperkuat konsep tersebut dengan properti bahwa evaluasi menuntut langkah komputasi sekuensial yang sulit dipercepat melalui paralelisme, namun menghasilkan keluaran yang dapat diverifikasi secara efisien (Boneh et al., 2018). Dua konstruksi VDF yang banyak dirujuk adalah skema Pietrzak dan Wesolowski yang menyediakan bukti ringkas serta verifikasi relatif murah dibanding biaya evaluasi, sehingga relevan untuk titik kontrol seperti *gateway edge* yang terbatas sumber daya (Pietrzak, 2019; Wesolowski, 2020). Survei VDF juga menempatkan VDF sebagai *primitive* yang sesuai untuk skenario penundaan terukur dan verifikasi publik pada sistem terdistribusi, termasuk aplikasi yang membutuhkan pembatasan laju berbasis kerja komputasi (Wu et al., 2022). Studi implementasi lebih lanjut menunjukkan

VDF dapat direalisasikan secara praktis dan kinerjanya dapat diukur, yang penting untuk menilai kelayakan penerapan pada lingkungan nyata (Attias et al., 2021).

Selain *primitive* kriptografis, pembentukan sinyal risiko memerlukan landasan dari pendekatan deteksi penyalahgunaan API. Deteksi berbasis pola perilaku memberikan kerangka untuk menilai anomali melalui telemetri seperti laju *request*, pola *burst*, dan kecenderungan *error*, yang dapat dipakai untuk pengambilan keputusan mitigasi secara cepat (Prinakaa et al., 2024). Perspektif lain menekankan bahwa kerentanan REST API dan penyalahgunaan sering berakar pada kelemahan kontrol akses dan proteksi trafik, sehingga kontrol di *gateway* perlu menutup celah yang bersifat operasional dan berulang (Tanveer et al., 2025). Pada saat yang sama, arah riset penambangan spesifikasi penggunaan API dari artefak kode menegaskan bahwa pemahaman terhadap pola penggunaan yang semestinya dapat memperkaya deteksi keamanan dan penilaian risiko (Yin et al., 2024). Dengan demikian, kombinasi telemetri *runtime* dan pemahaman perilaku penggunaan API menjadi dasar teoritis yang masuk akal untuk membentuk skor risiko ringan pada *gateway*.

Dalam konteks perbandingan pendekatan, beberapa penelitian mengeksplorasi *rate limiting* terdistribusi di sisi *load balancer* untuk menyebarkan beban kontrol dan mengurangi titik tunggal, tetapi tetap menghadapi tantangan koordinasi dan konsistensi kebijakan pada skala besar (Kalyanasundaram et al., 2025). Pendekatan lain mengusulkan *rate limiting* di sisi klien untuk mengurangi beban *server*, namun efektivitasnya bergantung pada kepatuhan klien dan tidak selalu kuat terhadap *adversary* yang memodifikasi perilaku klien (Farkiani et al., 2025). Kerangka teori tersebut memposisikan ARL-VDF sebagai pendekatan yang menggabungkan kontrol adaptif berbasis risiko dan "harga komputasi sekuensial" yang dapat diverifikasi, sehingga penyerang sulit mempertahankan *throughput* tinggi tanpa menaikkan biaya, sementara *gateway* tetap dapat memprioritaskan trafik berisiko rendah dalam batas sumber daya *edge* (Boneh et al., 2018; Pietrzak, 2019; Wesolowski, 2020).

3. METODE PENELITIAN

Model sistem

Sistem terdiri atas tiga komponen: (1) klien sebagai pengirim permintaan API, (2) *gateway edge* ringan sebagai titik kontrol (autentikasi, *routing*, *rate limiting*, dan verifikasi bukti), serta (3) *backend service* sebagai pemroses logika aplikasi. *Gateway* berperan sebagai *policy enforcement point* yang memutuskan apakah permintaan diteruskan ke *backend* atau ditolak/ditunda melalui mekanisme ARL-VDF. *Gateway* diasumsikan memiliki keterbatasan sumber daya (CPU dan memori) sehingga desain mekanisme harus menjaga agar *overhead*

server-side tetap rendah. Untuk itu, ARL-VDF memindahkan biaya utama mitigasi ke sisi klien melalui komputasi sekuensial (VDF), sementara *gateway* hanya melakukan verifikasi cepat.

Model Ancaman

Penyerang diasumsikan mampu: (i) mengirim trafik tinggi secara terdistribusi (multi identitas/bot), (ii) melakukan rotasi identitas/token untuk menghindari pembatasan berbasis identitas, serta (iii) mengeksploitasi *endpoint* mahal untuk memperbesar biaya pemrosesan. Penyerang juga diasumsikan dapat meniru perilaku permintaan sah hingga taraf tertentu (misalnya pola *burst*). Namun, penyerang tidak diasumsikan mampu mempercepat evaluasi VDF secara berarti melalui paralelisme karena VDF memerlukan langkah komputasi sekuensial, sehingga biaya waktu tetap melekat pada setiap bukti yang dihasilkan (Boneh et al., 2018; Pietrzak, 2019; Wesolowski, 2020). Asumsi ini menjadi dasar mengapa ARL-VDF dapat menekan laju efektif penyerang tanpa menaikkan beban *gateway* secara proporsional.

Telemetri permintaan dan pembentukan fitur

Gateway mengumpulkan telemetri per identitas i pada waktu t menggunakan jendela waktu tetap Δ atau pemulusan eksponensial (*exponential moving average*, EMA). Variabel utama yang dipakai adalah: laju permintaan $\hat{r}_i(t)$, rasio galat $\hat{e}_i(t)$, dan indikator identitas baru $n_i(t)$. Berikut adalah formula estimasi laju permintaan menggunakan jendela waktu (EMA):

$$\hat{r}_i(t) = \alpha \hat{r}_i(t - \Delta) + (1 - \alpha) r_i(t), 0 < \alpha < 1 \quad (1)$$

dengan $r_i(t)$ adalah laju permintaan teramati pada interval $[t - \Delta, t]$.

Rasio galat per identitas *i* pada waktu *t* didefinisikan sebagai proporsi respons gagal (misalnya HTTP 4xx/5xx) terhadap total respons pada interval pengamatan terakhir. Agar tidak sensitif terhadap fluktuasi sesaat, estimasi rasio galat dihitung menggunakan pemulusan eksponensial (*exponential moving average*, EMA). Jika $e_i(t) \in [0,1]$ menyatakan rasio galat teramati pada interval $[t - \Delta, t]$, maka estimasi *smoothed* $\hat{e}_i(t)$ diperbarui sebagai berikut.

$$\hat{e}_i(t) = \alpha \hat{e}_i(t - \Delta) + (1 - \alpha) e_i(t), \ 0 < \alpha < 1 \quad (2)$$

Dengan formulasi ini, nilai $\hat{e}_i(t)$ merepresentasikan tingkat kegagalan yang lebih stabil secara temporal, sehingga lebih sesuai untuk digunakan sebagai sinyal risiko daripada rasio mentah yang bisa dipengaruhi anomali sesaat.

Selain laju dan galat, sistem memerlukan sinyal yang menangkap apakah suatu identitas "baru muncul" pada horizon waktu tertentu, karena identitas baru yang tiba-tiba aktif sering berkorelasi dengan rotasi token atau pembuatan identitas massal. Indikator identitas baru didefinisikan sebagai variabel biner $n_i(t) \in \{0,1\}$ yang bernilai 1 apabila identitas *i* tidak teramati dalam horizon historis sepanjang *H* sebelum waktu *t*, dan bernilai 0 jika identitas

tersebut sudah pernah muncul pada horizon tersebut. Definisi formalnya dinyatakan sebagai berikut.

$$n_i(t) = \begin{cases} 1, Jika\ identitas\ i\ tidak\ teramati\ pada\ interval\ (t-H,t] \\ 0, Jika\ identitas\ i\ pernah\ teramati\ pada\ interval\ (t-H,t] \end{cases} \quad (3)$$

Untuk implementasi, *gateway* cukup menyimpan *timestamp* kemunculan terakhir per identitas; kemudian $n_i(t)$ ditetapkan berdasarkan apakah selisih waktu sejak kemunculan terakhir melebihi *H*. Pendefinisian ini membuat sinyal "baru terlihat" dapat dihitung ringan dan konsisten dengan karakteristik *gateway edge* yang terbatas.

Skor risiko dan pemetaan ke kesulitan VDF

Skor risiko dirumuskan sebagai probabilitas terikat [0,1] menggunakan fungsi logistik:

$$s_i(t) = \sigma\left(w_0 + w_1 z\left(\hat{r}_i(t)\right) + w_2 \hat{e}_i(t) + w_3 n_i(t)\right),\ \sigma(x) = 1/(1+e^{-}x) \quad (4)$$

Bobot $w_0, \ldots, w_3$ menyatakan kontribusi relatif tiap fitur. Dalam implementasi, bobot dapat ditetapkan (i) berbasis heuristik kebijakan keamanan, atau (ii) melalui kalibrasi awal menggunakan data log yang dilabeli (sah vs anomali). Skor $s_i(t)$ kemudian dipakai untuk menentukan tingkat kesulitan VDF dan besaran penundaan kriptografis.

Kesulitan VDF dipilih adaptif pada rentang $[D_{min}, D_{max}]$:

$$D_i(t) = D_{min} + s_i(t)(D_{max} - D_{min}) \quad (5)$$

Target penundaan dinyatakan sebagai estimasi waktu evaluasi sekuensial pada perangkat referensi:

$$\tau_i(t) = \frac{D_i(t)}{f_{\text{ref}}} \quad (6)$$

Dengan f_{ref} adalah kalibrasi langkah sekuensial per detik untuk platform klien (ditetapkan pada eksperimen/simulasi). Skema VDF mengikuti model *Setup*, *Eval*, *Verify* (Boneh et al., 2018), dengan bukti verifikasi efisien seperti pada konstruksi Pietrzak/Wesolowski (Pietrzak, 2019; Wesolowski, 2020). Pada tingkat sistem, parameter $\tau_i(t)$ diperlakukan sebagai "harga waktu" untuk permintaan berisiko sehingga laju efektif klien berisiko dibatasi oleh kebutuhan menghasilkan bukti.

Protokol tantangan VDF dan pengikatan ke permintaan

Agar bukti tidak dapat digunakan ulang (*replay*) atau dipindahkan antar permintaan, *gateway* menerbitkan tantangan yang mengikat konteks permintaan. Tantangan untuk identitas *i* pada waktu *t* dibangun dari *nonce* acak *u*, stempel waktu *t*, dan ringkasan permintaan *h(req)* (misal *hash path*+metode+parameter penting).

$$\text{chal} = H\left(u \,||t\, ||i\, ||h(req)\right) \quad (7)$$

Klien menjalankan $\mathrm{Eval}(\mathrm{chal}, D_i(t))$untuk memperoleh keluaran *y* dan bukti π. *Gateway* kemudian memeriksa $\mathrm{Verify}(\mathrm{chal}, y, \pi)$sebelum permintaan diteruskan. Mekanisme ini memastikan bukti hanya berlaku untuk permintaan tertentu dan periode tertentu, sehingga menekan peluang penyalahgunaan bukti yang sudah dihitung.

Aturan keputusan ARL-VDF pada *gateway*

ARL-VDF menggabungkan pembatasan laju ringan dan tantangan VDF selektif untuk menjaga trafik normal tetap lancar sekaligus menekan trafik berisiko. Gateway memelihara token bucket dasar per identitas dengan laju pengisian L_{base} dan kapasitas *B*; ketika skor risiko rendah $(s_i(t) < \theta)$ dan token masih tersedia, permintaan diterima tanpa memicu VDF agar latensi tetap minimal.

Jika token habis atau skor risiko meningkat $(s_i(t) \geq \theta)$, *gateway* tidak langsung menolak permintaan, melainkan menerbitkan tantangan VDF dengan parameter kesulitan $D_i(t)$atau target penundaan $\tau_i(t)$; permintaan hanya diteruskan ke *backend* bila klien mengembalikan keluaran dan bukti yang valid serta masih berada dalam masa berlaku tantangan. Permintaan dengan bukti tidak valid atau kedaluwarsa ditolak untuk mencegah *replay* dan penyalahgunaan.

Untuk mempertahankan sifat ringan pada perangkat *edge*, verifikasi bukti dibatasi oleh anggaran Γ verifikasi per detik; ketika anggaran terlampaui, permintaan yang membutuhkan verifikasi diprioritaskan sesuai kebijakan antrean yang ditetapkan (misalnya *first-come first-served* atau prioritas layanan), sedangkan sisanya ditunda atau ditolak, sehingga mekanisme mitigasi tidak menggeser *bottleneck* dari pemrosesan API ke verifikasi kriptografis pada *gateway*.

Dengan desain ini, trafik sah cenderung melewati jalur token *bucket* (tanpa VDF), sementara trafik berisiko tinggi dipaksa "membayar" biaya sekuensial pada sisi klien. Pada serangan terdistribusi, total biaya penyerang meningkat karena setiap bot harus menghitung bukti sendiri, sedangkan *gateway* hanya memverifikasi hingga batas anggaran.

Kompleksitas dan *overhead*

Overhead ARL-VDF didesain asimetris. Pada *gateway*, biaya utama adalah (i) pembaruan EMA dan evaluasi fungsi logistik (biaya konstan per permintaan), serta (ii) verifikasi VDF yang dibatasi anggaran Γ. Pada sisi klien berisiko tinggi, biaya dominan adalah evaluasi VDF yang sekuensial sebesar $D_i(t)$langkah, sehingga laju efektifnya dibatasi oleh $\tau_i(t)$. Dengan pembatasan Γ, *gateway* tidak dipaksa memverifikasi secara linear terhadap trafik

masuk pada saat serangan, dan tetap dapat menjaga ketersediaan layanan untuk jalur risiko rendah.

Desain evaluasi dan metrik

Evaluasi dilakukan melalui simulasi peristiwa diskrit selama 600 detik untuk memodelkan perilaku *gateway edge* pada kondisi beban campuran, yaitu trafik normal, *burst* trafik sah, dan trafik penyerang terdistribusi. Parameter dan konfigurasi simulasi dirangkum pada Tabel 1. Skenario normal merepresentasikan beban stabil dari klien sah, sedangkan burst trafik sah menggambarkan kondisi lonjakan periodik (misalnya sinkronisasi data atau *refresh* token) yang masih realistis pada aplikasi berbasis API. Trafik penyerang memodelkan bot yang menghasilkan permintaan secara paralel dengan tujuan menghabiskan kapasitas layanan atau memicu penolakan pada pengguna sah. *Baseline* pembanding adalah token *bucket* statis per identitas, sejalan praktik *rate limiting* yang banyak diadopsi pada API *gateway* (Bartkov & Borovikov, 2022; El Malki et al., 2022).

Tabel 1. Parameter simulasi.

Parameter	Nilai
Durasi simulasi	600 detik
Jumlah klien sah	300
Laju normal per klien sah	0.1 request/detik
Burst klien sah	5% klien, 8 request/detik selama 60 detik
Jumlah bot penyerang	30
Laju per bot penyerang	10 request/detik
Baseline limit token bucket (L_base)	5 request/detik per identitas
ARL-VDF tau_min	0.05 detik
ARL-VDF tau_max	0.60 detik
Anggaran verifikasi gateway	200 verifikasi/detik
Latensi dasar jaringan+backend	8 ms

Metrik evaluasi meliputi: (1) *success rate* trafik sah, yaitu proporsi permintaan sah yang berhasil diproses; (2) *throughput* serangan yang lolos, yaitu laju permintaan penyerang yang tetap diterima *gateway*; (3) latensi persentil-95 (p95) klien sah, untuk menangkap dampak terburuk yang masih dialami mayoritas pengguna; serta (4) estimasi utilisasi CPU verifikasi di *gateway*, yang dihitung dari jumlah verifikasi VDF per detik (dibatasi oleh anggaran verifikasi) dikalikan biaya verifikasi rata-rata per bukti. Pembatasan anggaran verifikasi pada *gateway* digunakan untuk menjaga agar mekanisme ARL-VDF tidak memindahkan *bottleneck* dari pemrosesan API menjadi *bottleneck* pada verifikasi kriptografis, sesuai karakteristik *gateway edge* yang memiliki sumber daya terbatas.

4. HASIL DAN PEMBAHASAN

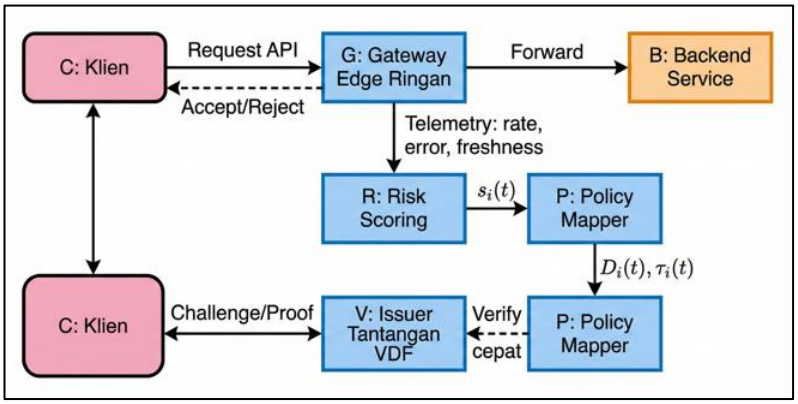
Penyajian data hasil

Kinerja *baseline* token *bucket* dan ARL-VDF dibandingkan menggunakan metrik agregat yang dirangkum pada Tabel 2. ARL-VDF meningkatkan *success rate* trafik sah dari 0,887 menjadi 1,000 dan menurunkan *throughput* penyerang yang lolos dari 148,9 menjadi 49,3 *request*/detik (Tabel 2). Dari sisi kualitas layanan, p95 latensi klien sah meningkat dari 8 ms menjadi 58 ms, sedangkan p95 latensi penyerang meningkat tajam hingga 608 ms (Tabel 2), yang menunjukkan friksi komputasi terutama dibebankan pada aliran berisiko tinggi.

Tabel 2. Ringkasan hasil simulasi.

Metrik	*Baseline* token *bucket*	ARL-VDF
Success rate trafik sah	0.887	1.000
Throughput penyerang yang lolos (request/detik)	148.9	49.3
p95 latensi klien sah (ms)	8	58
p95 latensi penyerang (ms)	8	608
Rata-rata verifikasi VDF (per detik)	0	56
Estimasi utilisasi CPU verifikasi (1 core, 3 ms/verify)	0.00	0.16

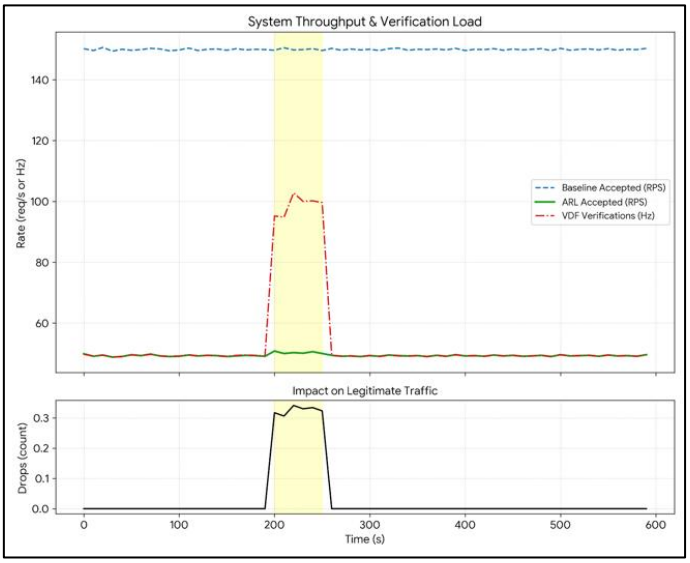
Pola hasil pada Tabel 2 dapat dijelaskan melalui alur modul ARL-VDF di Gambar 1. *Gateway* menghitung *risk scoring* dari telemetri permintaan, memetakan risiko ke parameter kebijakan $(D_i(t), \tau_i(t))$, lalu menerapkan *challenge/proof* VDF pada trafik berisiko (Gambar 1). Skema ini menahan biaya utama di sisi klien (evaluasi VDF) sementara *gateway* hanya melakukan verifikasi cepat, yang konsisten dengan overhead verifikasi yang tetap moderat pada Tabel 2.



Gambar 1. Arsitektur ARL-VDF pada *gateway edge*.

Selain metrik agregat, dinamika temporal selama simulasi disajikan pada Gambar 2. Data deret waktu pada Gambar 2 menunjukkan *throughput* penyerang pada baseline stabil di sekitar 149–150 rps, sedangkan ARL-VDF stabil di sekitar 49–50 rps pada sebagian besar interval. Pada rentang *burst* (sekitar detik 200–250), Gambar 2 menunjukkan peningkatan *drop rate*

trafik sah pada *baseline* (misalnya 0,317 pada detik 200) dan peningkatan aktivitas verifikasi VDF pada ARL-VDF (sekitar 95–103 verifikasi/detik), yang mengindikasikan mekanisme adaptif memperketat tantangan pada fase risiko meningkat (Gambar 2).



Gambar 2. *Throughput* sistem dan *verification load*

Perbandingan dengan penelitian terkait

Hasil pada Tabel 2 menunjukkan *baseline* token *bucket* masih meloloskan *throughput* penyerang tinggi, yang konsisten dengan karakter pembatasan per-identitas yang mudah diparalelkan melalui banyak bot/identitas. Hal ini sejalan dengan temuan bahwa kebijakan rate limit dapat memengaruhi reliabilitas layanan bila konfigurasi tidak adaptif terhadap dinamika beban (Sanchez et al., 2018), dan variasi adopsi di praktik industri menjelaskan heterogenitas efektivitas *limiter* statis (Bartkov & Borovikov, 2022). Kestabilan penurunan *throughput* penyerang pada ARL-VDF yang terlihat pada Gambar 2 memperkuat bahwa penundaan kriptografis selektif dapat membatasi kemampuan penyerang mempertahankan laju, bukan sekadar menolak permintaan.

Dibanding pendekatan pemilihan algoritme *limiter* yang berfokus pada *trade-off* internal (El Malki et al., 2022), ARL-VDF menambahkan dimensi biaya komputasi yang diverifikasi di *gateway*, seperti alur *challenge/proof* pada Gambar 1. Dibanding mekanisme resiliensi klien terhadap limit (Serbout & others, 2023), ARL-VDF menempatkan mitigasi pada *gateway* dan mengarahkan friksi ke kelas trafik berisiko, tercermin pada p95 latensi penyerang yang meningkat pada Tabel 2. Berbeda dari *limiter* terdistribusi di *load balancer* (Kalyanasundaram et al., 2025) atau kebijakan *client-side* murni (Farkiani et al., 2025), ARL-VDF mempertahankan *enforcement* di *gateway* (lihat Gambar 1) dengan *overhead* verifikasi yang tetap terukur (Tabel 2).

Kaitan teori dan interpretasi

Secara teori, *puzzle* efektif untuk mitigasi DoS/*abuse* bila biaya utama berada pada klien/penyerang dan verifikasi *server* murah (Ali et al., 2020). Pola pada Tabel 2 (turunnya *throughput* penyerang dan naiknya p95 latensi penyerang) konsisten dengan prinsip tersebut. VDF memperkuat *puzzle* melalui evaluasi sekuensial dan verifikasi efisien (Boneh et al., 2018; Pietrzak, 2019; Wesolowski, 2020), sehingga cocok pada *gateway edge* yang terbatas. Hal ini selaras dengan *overhead* verifikasi yang tercatat pada Tabel 2 serta alur verifikasi cepat pada Gambar 1. Selain itu, dinamika *burst* pada *baseline* yang memunculkan *drop rate* trafik sah di Gambar 2 menunjukkan keterbatasan limiter statis dalam menjaga kualitas layanan saat tekanan meningkat, sedangkan ARL-VDF mengintensifkan verifikasi pada interval risiko tinggi (Gambar 2) untuk menjaga performa trafik sah (Tabel 2).

5. KESIMPULAN DAN SARAN

Penelitian ini menghasilkan ARL-VDF, yaitu pembatasan laju adaptif pada *gateway edge* ringan yang menerapkan tantangan *Verifiable Delay Function* secara selektif berdasarkan skor risiko permintaan untuk menekan penyalahgunaan API tanpa mengurangi layanan bagi pengguna sah. Pada skenario evaluasi simulasi yang digunakan, ARL-VDF mempertahankan *success rate* trafik sah hingga 1,000 serta menurunkan *throughput* penyerang yang lolos dari 148,9 menjadi 49,3 *request*/detik, dengan peningkatan latensi p95 klien sah yang tetap terkendali dan beban verifikasi *gateway* yang moderat. Secara praktis, pendekatan ini dapat diterapkan sebagai kontrol perlindungan API pada *gateway edge* perangkat terbatas untuk menjaga ketersediaan layanan ketika terjadi serangan terdistribusi maupun lonjakan beban, karena biaya utama mitigasi didorong ke sisi klien berisiko tinggi melalui kerja komputasi sekuensial yang dapat diverifikasi. Penelitian selanjutnya perlu memvalidasi performa pada prototipe *gateway* nyata, menguji sensitivitas parameter pada variasi perangkat klien dan pola trafik, serta mengembangkan strategi kalibrasi adaptif agar tetap adil dan stabil pada kondisi produksi.

DAFTAR REFERENSI

Ali, I., Caprolu, M., & Di Pietro, R. (2020). Foundations, properties, and security applications of puzzles: A survey. *ACM Computing Surveys*, *53*(4), Article 88. https://doi.org/10.1145/3396374

Attias, D., Vigneri, L., & Dimitrov, D. (2021). Implementation study of two verifiable delay functions. In *Proceedings of Tokenomics 2020* (pp. 1–10). https://doi.org/10.4230/OASIcs.Tokenomics.2020.9

Bartkov, M., & Borovikov, D. (2022). Selection of a suitable algorithm for the implementation of rate-limiter based on Bucket4j. *International Journal of Online and Biomedical Engineering (IJOE), 18*(4), 51–62. https://doi.org/10.3991/ijoe.v18i04.25641

Boneh, D., Bonneau, J., Bünz, B., & Fisch, B. (2018). Verifiable delay functions. In *Advances in cryptology – CRYPTO 2018* (pp. 757–788). Springer. https://doi.org/10.1007/978-3-319-96884-1_25

Chandramouli, R., Romero-Mariona, S., & McCarthy, A. (2019). *Security strategies for microservices-based application systems* (NIST Special Publication No. 800-204). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204

El Malki, A., Zdun, U., & Pautasso, C. (2022). Impact of API rate limit on reliability of microservices-based architectures. In *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)* (pp. 19–28). IEEE. https://doi.org/10.1109/SOSE55356.2022.00009

Farkiani, S., Sarramia, D., & Lau, H. C. W. (2025). Rethinking HTTP API rate limiting: Client-side approach. *Manuscript submitted for publication*.

Hossain, M. D., et al. (2023). The role of microservice approach in edge computing: Opportunities, challenges, and research directions. *ICT Express*, *9*(6), 1162–1182. https://doi.org/10.1016/j.icte.2023.06.006

Kalyanasundaram, T., Punith, B., & V, S. (2025). Load balancer filter-based approach to enable distributed API rate limiting. In *Proceedings of the 37th Conference of FRUCT Association* (pp. 75–85).

Morabito, R., Petrolo, R., Loscrì, V., & Mitton, N. (2019). Reprint of: LEGIoT: A lightweight edge gateway for the Internet of Things. *Future Generation Computer Systems, 92*, 1157–1171. https://doi.org/10.1016/j.future.2018.10.020

Pietrzak, K. (2019). Simple verifiable delay functions. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS 2019)* (Article 60, pp. 1–18). https://doi.org/10.4230/LIPIcs.ITCS.2019.60

Prinakaa, S., Bavanika, V., Sanjana, S., Srinivasan, S., & Sarasvathi, V. (2024). A real-time approach to detecting API abuses based on behavioral patterns. In *2024 8th International Conference on Cryptography, Security and Privacy (CSP)* (pp. 24–28). IEEE. https://doi.org/10.1109/CSP62567.2024.00012

Sanchez, B. A., et al. (2018). RestMule: Enabling resilient clients for remote APIs. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)* (pp. 537–541). https://doi.org/10.1145/3196398.3196405

Serbout, B., et al. (2023). A pattern collection for API rate limit adoption. In *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP '23)* (pp. 1–20). https://doi.org/10.1145/3628034.3628039

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

Tanveer, F., Houssein, E. H., & Hassanien, A. E. (2025). A survey on RESTful API vulnerability detection. *Computers, Materials & Continua*, *84*(3), 4223–4257. https://doi.org/10.32604/cmc.2025.067536

Wesolowski, B. (2020). Efficient verifiable delay functions. *Journal of Cryptology*, *33*, 2113–2147. https://doi.org/10.1007/s00145-020-09364-x

Wu, Q., Zhao, Y., Luo, Z., & Zhang, X. (2022). Verifiable delay function and its blockchain-related applications: A survey. *Sensors*, *22*(19), 7524. https://doi.org/10.3390/s22197524

Xu, R., Jin, W., & Kim, D. (2019). Microservice security agent based on API gateway in edge computing. *Sensors*, *19*(22), 4905. https://doi.org/10.3390/s19224905

Yin, Z., Song, Y., & Zong, G. (2024). Discovering API usage specifications for security detection using two-stage code mining. *Cybersecurity*, *7*, Article 30. https://doi.org/10.1186/s42400-024-00224-w