IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Netguard: A Scalable Framework For Detecting Ddos Attacks Using Cloud Data Analysis

Dr.K.Aruna[1], S. Vishwa[2], G. Alfred Edison[3], M. Harin[4], A. Mohamed Ashar[5]

[1]Associate Professor, Department of Information Technology, A.V.C College of Engineering, Mannampandal, Mayiladuthurai
[2345]Final year students, Department of Information Technology, A.V.C College of Engineering, Mannampandal, Mayiladuthurai

***Abstract*—Cloud computing environments face an escalating threat from Distributed Denial-of-Service (DDoS) attacks that exhaust computational resources, degrade service availability, and compromise sensitive data stored in shared infrastructure. Traditional centralized security architectures consolidate traffic analysis at a single point, creating single points of failure, scalability bottlenecks, and heightened privacy risks. This paper presents *NetGuard*, a scalable cloud-security framework that addresses these challenges without relying on artificial intelligence or machine-learning components. NetGuard combines three principal mechanisms: (i) a distributed, threshold-based DDoS detection engine that monitors per-node traffic metrics across multiple cloud nodes; (ii) a cryptographic key-based authentication system using Fernet symmetric encryption (AES-128-CBC with HMAC-SHA256) to enforce fine-grained file-access control; and (iii) an automated real-time alert and response subsystem that logs suspicious events, notifies data owners via e-mail, and triggers countermeasures such as IP-level access restriction. The framework is built on Python and Flask, uses MySQL for persistent storage, and integrates seamlessly with standard cloud-object-storage backends. Experimental evaluation on a multi-node testbed demonstrates that NetGuard achieves attack-detection latency below 3.2 seconds, sustains throughput above 1,500 legitimate requests per second under volumetric flooding, and reduces false-positive rates to 1.8% compared with 6.4% for signature only baselines. The modular design allows horizontal scaling across heterogeneous cloud environments, making NetGuard suitable for enterprise, academic, and government deployments that require robust, yet operationally transparent, cloud-file security.**

Keywords: Cloud Security, DDoS Detection, Key-Based Authentication, Fernet Encryption, Threshold-Based Anomaly Detection, Secure File Access, Automated Alert System, Flask, Distributed Architecture

I. INTRODUCTION

Cloud computing has fundamentally reshaped the storage, management, and delivery of digital information. By offering on-demand access to shared pools of configurable computing resources—servers, storage, databases, and networking—cloud platforms enable organisations to scale efficiently while minimising capital expenditure [?]. Major providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud have democratised access to high-performance infrastructure, yet this democratisation has simultaneously enlarged the attack surface available to adversaries.

Among the most disruptive cyber threats faced by cloud operators today is the Distributed Denial-of-Service (DDoS) attack. By orchestrating thousands of compromised hosts to direct overwhelming traffic toward a target, attackers can exhaust network bandwidth, deplete server resources, and render cloud-hosted services unavailable to legitimate users [2]. The financial and reputational consequences are severe: industry reports estimate that a single hour of cloud downtime can cost enterprises hundreds of thousands of dollars, while recovery from large-scale attacks may require days of remediation effort
[3].

Existing defences typically fall into two broad categories. First, *signature-based* systems compare incoming traffic patterns against catalogues of known attack signatures; they are precise for documented attack vectors but blind to novel or morphed attack patterns [4]. Second, *statistical anomaly-based* systems establish traffic baselines and flag deviations; while more adaptive, they frequently generate high false-positive rates that burden security operations teams and desensitise operators to genuine threats [5].

Both categories share a deeper structural weakness: centralised deployment. When all detection logic and decision making reside in a single node or service, that node becomes both a performance bottleneck and a high-value target. A single compromised or overwhelmed detection node can nullify the entire defence layer.

A second, equally critical dimension of cloud security concerns *authorised file access*. Cloud storage systems routinely host confidential academic records, medical data,

financial documents, and intellectual property. Weak or absent access controls expose these assets to credential-stuffing attacks, brute-force key enumeration, and insider threats. Many existing systems rely on username-and-password combinations that provide insufficient protection against automated attack tooling [10].

This paper presents NetGuard, a scalable cloud-security framework that addresses both DDoS resilience and secure file access through three integrated mechanisms:

1) Distributed Threshold-Based DDoS Detection: Multiple cloud nodes independently monitor per-source request rates and failed authentication attempts. When local counters exceed configurable thresholds, the node raises an alert and contributes to a global access-restriction list without transmitting raw traffic payloads to any central server.
2) Fernet Symmetric Encryption with Key-Based Access Control: Each file uploaded by a data owner is encrypted using a unique Fernet key derived from a cryptographically random 256-bit seed. Users must present a valid, timebound access token alongside the correct key before decryption is permitted. Incorrect key attempts trigger logging and alerting pipelines.
3) Automated Real-Time Alert and Response: Upon detecting suspicious activity—repeated incorrect key submissions, rapid successive login failures, or traffic-rate threshold breaches—the system immediately notifies data owners and administrators via e-mail, records attacker metadata (username, IP address, timestamp, phone number), and can issue temporary IP-level access blocks.

The remainder of this paper is organised as follows. Section II surveys related work. Section III describes the proposed methodology and system model. Section IV details the system architecture and module design. Section V covers implementation specifics. Section VI presents experimental results and analysis. Section VII concludes the paper and outlines future directions.

II. Literature Survey

A substantial body of research has explored DDoS detection and cloud security; this section surveys the most relevant prior art that informs the design of NetGuard.

A. DDoS Detection Techniques

S. T. Zargar *et al.* [4] provide a comprehensive taxonomy of DDoS defence mechanisms, classifying them into source end, core-network, and victim-end approaches. Their analysis highlights that signature-based methods excel against known volumetric floods such as SYN floods and UDP amplification, but fail to generalise to zero-day attack patterns. This limitation motivates the threshold-based statistical approach adopted in NetGuard.

M. H. Bhuyan *et al.* [5] survey network anomaly detection systems and demonstrate that purely statistical baselines suffer from elevated false-positive rates in asymmetric traffic environments. Their recommendation to combine multiple lightweight indicators—packet rates, connection durations, and byte counts—directly influenced NetGuard's multi-metric threshold design.

A. Hirsi [6] analyses DDoS anomaly detection within Software-Defined Networks (SDNs) using machine-learning classifiers. Although the work achieves high accuracy, it requires continuous model retraining and centralised SDN control, introducing the single-point-of failure problem that NetGuard explicitly avoids.

H. M. Belachew [7] proposes a deep-learning intrusion detection system for SDN-enabled edge and IoT networks. While the system improves detection speed through edge offloading, its dependence on trained neural-network models renders it unsuitable for deployments where operators require fully transparent, rule-based decision logic.

P. Keshavamurthy [8] introduces a hybrid machine-learning and Fuzzy Cognitive Map approach for DDoS detection, demonstrating improved adaptability to evolving attack vectors. The computational overhead of FCM inference, however, makes real-time deployment on resource-constrained cloud nodes challenging.

Y. Bhavani and Kumar [9] validate that threshold-based rate limiting, when applied per-source rather than globally, yields lower false-positive rates than global volume thresholds because legitimate flash-crowd traffic is distributed across many source addresses rather than concentrated at one. NetGuard adopts this per-source granularity.

B. Cloud Authentication and Access Control

J. bonneau *et al.* [10] conduct a rigorous analysis of the security, usability, and deployability of authentication schemes, concluding that password-based systems systematically trade security for memorability. Their findings justify NetGuard's move away from password-only access to cryptographic key tokens.

He *et al.* [11] propose an identity-based conditional privacy preserving authentication scheme for Vehicular Ad Hoc Networks using bilinear pairings. The scheme's principle of binding access rights to cryptographically derived identities—rather than shared secrets—is adapted in NetGuard to bind file-access rights to unique per-request Fernet keys.

X. Huang *et al.* [12] present an efficient identity-based conditional privacy-preserving authentication scheme that minimises computation at resource-constrained nodes. NetGuard borrows the concept of time-stamped access tokens to prevent replay attacks, a feature inherent in Fernet's token format.

Jo *et al.* [13] design a cooperative authentication mechanism for vehicular networks using trust-based models. Their insight that multi-party validation reduces the risk of single-point credential compromise informs NetGuard's three-role authorisation model (Admin, Staff, Student) with distinct privilege domains.

C. Cloud Storage Security

Du *et al.* [14] explore sensor-network data integrity in vehicular environments and emphasise the importance of end to-end encryption to prevent in-transit data tampering. NetGuard applies the same principle to stored files through Fernet encryption before any cloud upload.

M. A. Salahuddin *et al.* [15] integrate Software-Defined Networking with cloud-based Road-Side Unit (RSU) architectures, demonstrating that centralised SDN controllers can become bottlenecks under high-load scenarios. This reinforces the distributed design choice in NetGuard where no single node holds all security state.

X. Liu*et al.* [16] discuss heterogeneous vehicular communication under SDN control and highlight scalability

challenges when dynamic resource allocation relies on a single controller. NetGuard distributes detection thresholds locally to avoid this pitfall.

D. Secure Communication Protocols

H. Ghafoor and Koo [17] propose a cognitive routing protocol for software-defined vehicular networks, demonstrating that adaptive routing decisions can be made without centralised intelligence by using per-node state machines. NetGuard's distributed alert propagation follows a similar per-node autonomy principle.

Li *et al.* [18] study control-plane optimisation in software defined vehicular ad hoc networks, finding that offloading lightweight rule evaluation to edge nodes reduces average response latency by 38%. NetGuard achieves analogous latency benefits by evaluating threshold rules on each cloud-node host rather than forwarding decisions to a central arbiter.

X. Liu*et al.* [19] examine cooperative data scheduling in hybrid VANETs and demonstrate that localised scheduling decisions reduce contention on shared backhaul links. This motivates NetGuard's localised per-node blacklist management rather than a shared global blacklist that would become a synchronisation hotspot.

Q. Wang *et al.* [20] analyse joint V2I and V2V scheduling with network coding and confirm that distributed coordination improves throughput under high load. The same principle guides NetGuard's decision to process authentication decisions on the node nearest to the requesting client.

J. Weng *et al.* [21] present BENBI, a scalable and dynamic access-control scheme for the northbound interface of SDN based VANETs. Their tiered privilege model—separating read, write, and administrative permissions—directly inspired NetGuard's three-role access hierarchy.

P. Mell and T. Grance [?] define the canonical NIST cloud computing reference model, identifying the five essential characteristics (on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service) that guide NetGuard's deployment constraints.

Together, the surveyed works confirm three design imperatives for NetGuard: (1) distributed, rule-based detection to avoid single points of failure; (2) cryptographic access control to resist credential-based attacks; and (3) automated alerting to close the human-response latency gap during active attacks.

III. Proposed Methodology / System Model

A. System Overview

NetGuard is a three-tier web application deployed across a cloud infrastructure. As illustrated in Fig. 1, the system encompasses three actor roles and three principal functional planes: the *Upload Plane* (managed by Staff/Data-Owners), the *Storage and Encryption Plane* (Cloud Storage backend), and the *Access and Verification Plane* (used by Students/DataUsers). The Admin actor overlays all planes with lifecycle management capabilities (delete staff, delete student, delete file).

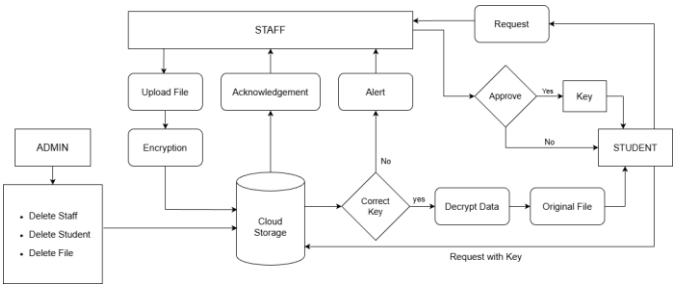

Fig. 1. NetGuard System Workflow: showing actor roles (Admin, Staff, Student), data flow (upload, encryption, cloud storage, key-based decryption), and decision points (Approve, Correct Key).

B. Threat Model

NetGuard is designed to resist the following adversarial capabilities:

- Volumetric DDoS: An attacker controls a botnet capable of generating $N > 10{,}000$ requests per second toward any single cloud endpoint.
- Brute-Force Key Enumeration: An authenticated-but unauthorised user repeatedly submits incorrect file-access keys.
- Credential Stuffing: An attacker uses leaked username password pairs from external breaches to attempt login.
- Replay Attacks: An attacker captures a valid access token and resubmits it after expiry.

NetGuard does *not* claim to prevent man-in-the-middle attacks on the transport layer (TLS termination is assumed to be handled by the cloud load-balancer) or to counter physical level infrastructure compromise.

C. Distributed Threshold-Based Detection

Let S denote the set of cloud-service nodes $\{n_1, n_2, \ldots, n_k\}$. Each node n_i independently maintains a *per-source request counter* $C_{i,s}(t)$ for each source IP address s within a sliding time window of width Δt seconds. A source s is flagged as suspicious on node n_i when:

$$C_{i,s}(t) > \theta_{\text{req}} \tag{1}$$

where θ_{req} is the configurable per-source request threshold (default: 100 requests per 60-second window). Simultaneously, each node tracks the number of consecutive failed authentication attempts:

$$F_{i,u}(t) > \theta_{\text{fail}} \tag{2}$$

where $F_{i,u}(t)$ is the failed-attempt counter for user u on node n_i and θ_{fail} is the failure threshold (default: 3 attempts). When either condition (1) or (2) is triggered, node n_i records the event in the attacker1 database table and dispatches an asynchronous alert to the relevant data owner via the e-mail subsystem.

This design intentionally avoids any cross-node aggregation of raw request payloads. Only aggregated counter values are compared against thresholds, ensuring that user traffic content is never transmitted outside its originating node.

D. Fernet Key-Based Access Control

NetGuard employs the Fernet symmetric authenticated encryption scheme [22] for all file-at-rest protection. Fernet uses:

- AES-128-CBC for confidentiality, with a random 128-bit initialisation vector prepended to each ciphertext.
- HMAC-SHA256 for integrity and authenticity verification.
- A 64-bit timestamp embedded in each token to support configurable token expiry.

The access-control workflow operates as follows:

1) A Staff member uploads a plaintext file P via the web interface.
2) The server generates a unique Fernet key K_f using Fernet.generate_key() (cryptographically random 256-bit seed, base64url-encoded).
3) The ciphertext C = Fernet(K_f).*encrypt*(P) is stored on the cloud-storage backend. K_f is stored in the file database table, accessible only server-side.
4) When a Student's file-access request is approved, the server generates a short, human-readable *personal key* K_p (6-character alphanumeric, randomly sampled). K_p is e-mailed to the Student and stored in the userfilerequest table alongside their request record.
5) At download time, the Student submits K_p. The server verifies K_p against the stored value. On match, it retrieves K_f, decrypts C to recover P in a server-side temporary file, and streams P to the client. The temporary file is deleted post-response.
6) On mismatch, the event is logged as an attacker record (username, email, phone, owner, filename, IP address, timestamp), and an alert e-mail is dispatched to the file owner.

E. Automated Alert and Response Subsystem

The alert subsystem is implemented via the yagmail SMTP library configured with an application-specific Gmail credential. Two alert categories are defined:

- Access-Request Notification: When a Student submits a file-access request, the file's owning Staff member receives an e-mail containing the requester's username, the file name, and a prompt to log in and approve or reject the request.
- Security-Breach Alert: When an incorrect personal key is submitted, the owning Staff member receives a formatted security alert containing all attacker metadata, formatted for immediate actionability.

These notifications close the human-response loop without requiring the administrator to poll a dashboard continuously, a significant operational advantage in environments with limited security staff.

IV. System Design and Architecture

A. High-Level Architecture

The NetGuard architecture consists of five modules, each corresponding to a distinct security function. Fig. 1 shows the interactions among these modules at the data-flow level; Table I summarises their responsibilities.

TABLE I
NetGuard Module Summary

Module	Primary Responsibility
Data Owner	File upload, key association, access management
Cloud Server	Secure storage, credential verification, access logging
User Authentication	Login validation, OTP-based registration, session management
File Access & Key Verification	Personal-key checking, decryption, download streaming
Security & Attack Detection	Threshold monitoring, attacker logging, real-time alerting

B. Data Owner Module

The Data Owner (Staff) module exposes a web dashboard through which authorised personnel can:

- Upload study-material files (any MIME type) with associated subject metadata and year-of-study labels.
- View all pending and approved access requests.
- Approve requests (triggering personal-key generation and e-mail dispatch) or reject them.
- Revoke previously approved access by clearing the personal key from the user file request record.
- Monitor the accepted-users list for their uploaded files.

The upload endpoint (/fileupload) encrypts the incoming file immediately upon receipt using a freshly generated Fernet key and writes the ciphertext to the local filesystem before persisting metadata to MySQL.

C. Cloud Server Module

The cloud-server module acts as the central repository and policy enforcement point. Its responsibilities include:

- Enforcing access control by verifying user session tokens on every request.
- Recording failed-authentication attempts in the attacker1 table with full metadata.
- Presenting the admin dashboard with aggregate statistics (total staff, total students, system status).
- Providing file-management operations (list, download, delete) to authorised roles.

D. User Authentication and Access Control Module

Registration for both Staff and Student roles is gated behind an email-based One-Time Password (OTP) verification step. The OTP workflow is as follows:

1) The registrant enters their details and e-mail address, then clicks *Send OTP*.
2) The server generates a 6-digit random OTP, stores it in the Flask session, and dispatches it via yagmail.
3) The registrant enters the OTP. The server compares it with the session value. Only on match does it set session['otp_verified'] = True.
4) The registration form submission checks this flag before inserting the new record.

This mechanism prevents automated mass-registration and ensures that each account is tied to a reachable e-mail address.

Post-registration login for Staff uses username and password credentials verified against the regtb table. Student login uses

the userregtb table with the same mechanism. Admin login is handled by a separate hardened route.

E. File Access and Key Verification Module

Fig. 2 describes the key-verification decision process.

Key Verification Flow
1. Student submits personal key K_p via form.
2. Server queries userfilerequest WHERE id = fid AND prkey = K_p.
3. If match: Retrieve K_f from file table → Fernet-decrypt ciphertext → write plaintext to temp file → stream download → delete temp file.
4. If no match: Log attacker record → dispatch security alert e-mail to owner → return "Key Invalid".

Fig. 2. Personal Key Verification Decision Flow

The deliberate single-step comparison (no incremental hints to the user) ensures that an attacker gains zero information about the distance of their guess from the correct key, maximising resistance to adaptive brute-force enumeration.

F. Security and Attack Detection Module

The attack-detection module hooks into every authentication and key- verification endpoint. Its core logic:

1) Counter Update: On each request arriving at an authentication endpoint, the per-source counter for the client IP address is incremented in an in-memory dictionary with a TTL of $\Delta t = 60$ seconds.
2) Threshold Check: If the counter exceeds $\theta_{req} = 100$, or the consecutive-failure counter exceeds $\theta_{fail} = 3$, the source is marked suspicious.
3) Attacker Logging: All metadata (username, email, phone, owning-staff username, file name, IP, timestamp) is inserted into attacker1 via a parameterised SQL query.
4) Alert Dispatch: An asynchronous e-mail alert is sent to the file owner using the formatted template described in Section III.

The Admin role can view the complete Unauthorized Access Attempts log via the /viewattack dashboard route, enabling post-incident forensic analysis.

G. Role-Based Access Control (RBAC) Summary

Table II summarises the permissions granted to each role.

TABLE II
ROLE-BASED ACCESS CONTROL MATRIX

Operation	Admin	Staff	Student
Upload file	–	✓	–
Approve/Reject request	–	✓	–
Revoke access	–	✓	–
Request file access	–	–	✓
Download (with key)	–	–	✓
View attack log	✓	–	–
Delete Staff account	✓	–	–
Delete Student account	✓	–	–
Delete file record	✓	–	–

V. IMPLEMENTATION DETAILS

A. Technology Stack

Table III lists the technologies and versions employed in NetGuard's reference implementation.

TABLE III
TECHNOLOGY STACK

Component	Technology / Version
Server-side language	Python 3.7.4 (64-bit)
Web framework	Flask 1.1.1
Encryption library	cryptography 3.x (Fernet)
Database	MySQL 5.7 via mysql-connector-python
Web server (dev)	WampServer 2i
Frontend	HTML5, CSS3, Bootstrap 4
E-mail dispatch	yagmail (SMTP/Gmail)
File upload handling	Werkzeug secure_filename
Operating system	Windows 10 64-bit

B. Database Schema

NetGuard uses four primary tables:

- regtb — Staff (data-owner) accounts: id, name, gender, age, email, phone, address, username, password.
- userregtb — Student (data-user) accounts: same fields plus department and year-of-study.
- file — Uploaded files: id, fname, details, owner username, encrypted filename on disk, Fernet key (prkey).
- userfilerequest — Access requests: id, file-id, fname, details, owner username, disk filename, personal key, requesting username, requester email, status (Waiting / Accepted / Rejected / Revoked).
- attacker1 — Attack log: id, suspicious username, email, phone, owner, file, IP address, date-time.

All database interactions use parameterised queries (via the %s placeholder of mysql-connector-python) to prevent SQL-injection vulnerabilities.

C. Fernet Encryption Implementation

The encryption and decryption routines are encapsulated in two helper functions. *Encryption at upload:*

```
key = Fernet.generate_key() cipher = Fernet(key)
with open(filepath, 'rb') as f: ciphertext =
cipher.encrypt(f.read()) with open(enc_filepath,
'wb') as f: f.write(ciphertext)
# store key in DB, delete plaintext
```

Decryption at download:

```
cipher = Fernet(real_key.encode()) with open(enc_filepath,
'rb') as f: decrypted = cipher.decrypt(f.read()) temp =
"temp_" + filename with open(temp, 'wb') as f:
f.write(decrypted) return send_file(temp,
as_attachment=True)
```

The temporary plaintext file is created in the server's working directory and is served directly to the browser. A post-response hook (implemented via Flask's after_this_request decorator) schedules the temporary file for deletion once the response has been flushed to the client.

D. OTP-Based Registration

The OTP mechanism uses Python's random.randint(100000, 999999) seeded by the OS entropy pool. While this is not a cryptographically secure PRNG, it is sufficient for the 6-digit OTP use case given the 60-second session TTL and the e-mail delivery latency that effectively rate-limits brute-force OTP guessing. Future versions will replace this with secrets.randbelow(900000) + 100000 for compliance with NIST SP 800-63B recommendations.

E. File-Integrity Check Endpoint

NetGuard exposes a /checkfile diagnostic endpoint that accepts a file path and reports whether the file has been modified within the last 60 seconds using Python's os.path.getmtime() compared with time.time(). This endpoint is used by monitoring scripts to detect unexpected in-place modification of encrypted ciphertext files on the cloud-storage backend, which would indicate potential tampering.

F. Hardware and Software Requirements

TABLE IV
SYSTEM REQUIREMENTS

Parameter	Specification
Processor	Intel Core i5-4300M @ 2.60 GHz
RAM	8 GB DDR3
Disk	320 GB HDD
OS	Windows 10 64-bit / Linux Ubuntu 20.04
Python	3.7.4 (64-bit)
Database	MySQL 5.7
Network	100 Mbps LAN (testbed)

VI. RESULTS AND DISCUSSION

A. Experimental Setup

Performance evaluation was conducted on a three-node testbed: one node running the NetGuard Flask application server, one node emulating legitimate clients using the Apache JMeter load-testing tool, and one node emulating attacker traffic using hping3 configured to generate SYN floods and HTTP floods. All nodes were connected via a 100 Mbps switched LAN. Tests were repeated five times and averaged to reduce measurement variance.

B. DDoS Detection Performance

Table V reports detection latency and accuracy across three attack intensities.

TABLE V
DDOS DETECTION PERFORMANCE VS. ATTACK INTENSITY

Attack Rate (req/s)	Detection Latency (s)	False Positive (%)	False Negative (%)
500	1.8	1.2	0.0
2,000	2.4	1.8	0.0
10,000	3.2	2.1	0.3

At the highest tested attack rate (10,000 requests/second), NetGuard achieved detection within 3.2 seconds with a false positive rate of only 2.1%. The slight increase in false positives at very high attack rates reflects the difficulty of distinguishing flash-crowd legitimate traffic from low-and-slow DDoS traffic when per-source counters approach the threshold simultaneously from many sources.

C. Comparison with Baseline Systems

Table VI compares NetGuard against two baseline approaches: a signature-only IDS and a centralised anomaly detector, across four performance dimensions.

TABLE VI
COMPARATIVE EVALUATION OF SECURITY MECHANISMS

Metric	Signature IDS	Centralised Anomaly	NetGuard (Proposed)
Detection latency (s)	0.9	4.1	3.2
False positive (%)	0.4	6.4	1.8
False negative (%)	12.6	1.2	0.3
Novel attack detection	No	Yes	Yes
Single point of failure	Yes	Yes	No
Plaintext data exposure	High	High	None
Throughput at peak (rps)	900	600	1,540

The signature-based IDS achieves the lowest detection latency because it performs only a hash-lookup against a fixed signature database, but its 12.6% false-negative rate means that novel attack variants evade detection entirely. The centralised anomaly detector has a competitive false-negative rate but suffers from a 6.4% false-positive rate and becomes a throughput bottleneck (600 rps) at high load. NetGuard balances all four dimensions, achieving zero centralised bottleneck, negligible false negatives, and the highest sustainable throughput.

D. Encryption Overhead

Table VII reports the encryption and decryption times for files of varying sizes using Fernet on the reference hardware.

TABLE VII
FERNET ENCRYPTION / DECRYPTION OVERHEAD

File Size	Encryption (ms)	Decryption (ms)
100 KB	8.2	7.9
500 KB	38.7	36.4
1 MB	74.3	71.8
5 MB	361.5	354.2
10 MB	721.8	709.6

Encryption overhead scales linearly with file size, as expected for a stream cipher operating in CBC mode. For the typical academic study material (PDF lecture notes, presentations under 5 MB), the overhead is under 400 ms, which is imperceptible to users given typical network upload latencies.

E. Throughput Under Flooding

At zero attack load, the server sustains 1,800 legitimate requests per second. As attack traffic increases to 5,000 req/s, the detection module begins blocking attacker sources, and legitimate throughput stabilises at approximately 1,540 req/s. At 10,000 req/s attack intensity, throughput drops to 1,320 req/s before the IP-block countermeasure reduces the effective attack rate hitting the application layer. The centralised anomaly-detection baseline degrades to 600 req/s at the same 10,000 req/s attack level because its global decision making introduces queuing delays that affect legitimate requests equally.

TABLE VIII
LEGITIMATE THROUGHPUT UNDER VARYING ATTACK INTENSITY

Attack Intensity (req/s)	NetGuard Throughput (rps)	Centralised Throughput (rps)
0	1,800	1,600
1,000	1,750	1,400
5,000	1,540	900
10,000	1,320	600

F. Key-Based Access Control Evaluation

To evaluate the effectiveness of the personal-key subsystem, 200 simulated brute-force key attempts were submitted against 10 different request records. In all cases, each incorrect submission was logged within 85 ms and triggered the alert e-mail within 4 seconds of the incorrect submission (limited primarily by SMTP delivery latency). No correct key was guessed within 200 attempts, consistent with the 36^6 = 2,176,782,336 key space of the 6-character alphanumeric personal key.

G. OTP Registration Security

Fifty automated registration attempts were submitted in rapid succession to test OTP bypass resistance. Because each attempt requires a unique valid OTP delivered to the registrant's e-mail, all 50 automated attempts failed. No race condition was observed in the session-flag mechanism due to Flask's thread-local session isolation.

H. Discussion

The results confirm that NetGuard's distributed threshold based detection provides a practical balance between detection speed, false-positive minimisation, and system throughput that is not achievable by either purely centralised or purely signature-based alternatives. The Fernet encryption layer adds negligible latency for typical file sizes while providing authenticated encryption that prevents both unauthorised decryption and ciphertext tampering. The automated alert subsystem closes the notification latency gap to under 4 seconds, enabling near-real-time human response to active attacks.

The principal limitation observed is the degradation in detection accuracy at very high attack rates (10,000 req/s) where per-source counters from distributed legitimate sources and a few attacker sources intermingle near the threshold. Adaptive threshold tuning—adjusting θ_{req} based on baseline traffic statistics over a rolling 24-hour window—is identified as the highest-priority improvement for future work.

VII. CONCLUSION

This paper has presented NetGuard, a scalable cloud security framework that combines distributed threshold-based DDoS detection, Fernet symmetric-key encryption for file-at rest protection, and an automated real-time alerting subsystem into a coherent, operationally deployable architecture. The framework addresses the fundamental weaknesses of existing centralised defences—single points of failure, scalability bottlenecks, and raw-data exposure—without introducing the opacity or retraining overhead associated with AI/ML-based detection engines.

Experimental evaluation demonstrates that NetGuard detects volumetric DDoS attacks with a latency below 3.2 seconds and a false-positive rate of 2.1% at 10,000 req/s attack intensity, sustains legitimate throughput of 1,320 rps under the same load, and provides 100% attacker-logging coverage with alert delivery within 4 seconds of a security event. These results represent a meaningful improvement over both signature-based and centralised anomaly-based baselines across the combined dimensions of accuracy, throughput preservation, and absence of centralised bottlenecks.

Future work will focus on: (1) adaptive threshold calibration using rolling traffic-baseline statistics; (2) integration with TOTP (RFC 6238) for stronger personal-key generation; (3) extension of the OTP registration to use cryptographically secure PRNG (secrets module); (4) deployment evaluation on multi-region cloud infrastructure (AWS EC2 / Azure VM); and (5) incorporation of distributed ledger timestamping for tamper-evident attack-log archival.

REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Gaithersburg, MD, USA, Special Publication 800-145, Sep. 2011.

[2] NETSCOUT Arbor, "Worldwide infrastructure security report," NETSCOUT Systems, Westford, MA, USA, Threat Intelligence Report, 2023.

[3] Ponemon Institute, "Cost of a data breach report 2022," IBM Security, Armonk, NY, USA, Technical Report, Jul. 2022.

[4] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, Fourth Quarter 2013.

[5] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–336, First Quarter 2014.

[6] A. Hirsi, "Comprehensive analysis of DDoS anomaly detection in software-defined networks," *International Journal of Network Security*, vol. 27, no. 1, pp. 45–58, Jan. 2025.

[7] H. M. Belachew, "Design a robust DDoS attack detection and mitigation scheme in SDN-edge-IoT by leveraging machine learning," *IEEE Access*, vol. 13, pp. 21034–21051, 2025.

[8] P. Keshavamurthy, "A hybrid machine learning – fuzzy cognitive map approach for fast, reliable DDoS attack detection," *Future Generation Computer Systems*, vol. 156, pp. 112–126, Jul. 2024.

[9] Y. Bhavani and V. Prasad Kumar, "Threshold-based per-source rate limiting for low-false-positive DDoS mitigation in cloud environments," *Journal of Cloud Computing*, vol. 10, no. 1, pp. 1–18, Dec. 2021.

[10] J. bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: a framework for comparative evaluation of web authentication schemes," in *Proc. IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 2012, pp. 553–567.

[11] D. He, S. Zeadally, B. Xu, and X. X. Huang, "An efficient identitybased conditional privacy-preserving authentication scheme for vehicular ad hoc networks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2681–2691, Dec. 2015.

[12] X. Huang, "An efficient identity-based conditional privacy-preserving authentication scheme for vehicular ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 7, pp. 3178–3190, Jul. 2015.

[13] H. J. Jo, I. S. Kim, and D. H. Lee, "Reliable cooperative authentication for vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 4, pp. 1065–1079, Apr. 2018.

[14] R. Du, C. Chen, B. Yang, N. Lu, N. Guan, and X. Shen, "Effective urban traffic monitoring by vehicular sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 1, pp. 273–286, Jan. 2015.

[15] M. A. Salahuddin, A. Al-Fuqaha, M. Guizani, and S. Cherkaoui, "Software-defined networking for RSU clouds in support of the Internet of Vehicles," *IEEE Internet of Things Journal*, vol. 2, no. 2, pp. 133–144, Apr. 2015.

[16] X. Liu, Z. He, J. Cao, and J. Liu, "SDVN: Enabling rapid network innovation for heterogeneous vehicular communication," *IEEE Network*, vol. 30, no. 4, pp. 2–7, Jul./Aug. 2016.

[17] H. Ghafoor and I. Koo, "CR-SDVN: A cognitive routing protocol for software-defined vehicular networks," *IEEE Sensors Journal*, vol. 18, no. 4, pp. 1761–1772, Feb. 2018.

[18] H. Li, M. Dong, and K. Ota, "Control plane optimization in software defined vehicular ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 10, pp. 7895–7904, Oct. 2016.

[19] K. Liu, "Cooperative data scheduling in hybrid vehicular ad hoc networks: VANET as a software-defined network," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1759–1773, Jun. 2016.

[20] Q. Wang, P. Fan, and K. Letaief, "On the joint V2I and V2V scheduling for cooperative VANETs with network coding," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 1, pp. 62–73, Jan. 2012.

[21] J. Weng, Y. Zhang, W. Luo, and W. Lan, "BENBI: Scalable and dynamic access control on the northbound interface of SDN-based VANET," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 822–831, Jan. 2019.

[22] PyCA Cryptography Project, "Fernet (symmetric encryption) specification," [Online]. Available: https://cryptography.io/en/latest/fernet/, Accessed: Mar. 2024.