INTEGRITY VERIFICATION OF WEBSITE CACHE USING HIERARCHICAL MAC CONSTRUCTIONS FOR EDGE COMPUTING PARADIGMS

XÁC MINH TÍNH TOÀN VẸN CỦA BỘ NHỚ ĐỆM WEBSITE SỬ DỤNG CẤU TRÚC MAC PHÂN CẤP CHO CÁC MÔ HÌNH ĐIỆN TOÁN BIÊN

Quoc Khanh Trinh, Nguyen Van Nguyen, Nguyen Nang Hung Van*, Pham Minh Tuan

The University of Danang - University of Science and Technology, Vietnam

*Corresponding author: nguyenvan@dut.udn.vn

(Received: March 27, 2026; Revised: May 13, 2026; Accepted: May 21, 2026)

DOI: 10.31130/ud-jst.2026.24(5A).196E

Abstract - This paper presents a hierarchical Message Authentication Code (MAC) framework for ensuring the integrity of web caches in edge-computing environments. The system introduces three coordinated verification layers: content integrity checking, node authentication using dynamic trust scores, and system-wide consistency validation. We formalize each layer using HMAC-SHA256 with a structured key-derivation process and incorporate exponential trust-decay and recovery models for adaptive security. A proof-of-concept implementation shows that the approach offers strong resistance to tampering and impersonation while maintaining low computational overhead. Experimental evaluation demonstrates that hierarchical MACs not only enhance security but also support scalability, robustness against diverse attack scenarios, and efficient real-time verification. The results show that hierarchical MACs provide a practical and scalable solution for securing website cache infrastructures.

Key words - Web applications security; Web cache integrity; Edge computing; Hierarchical authentication

Tóm tắt - Bài báo đề xuất khung xác thực mã thông điệp (MAC) phân cấp cho bộ nhớ đệm web trong môi trường điện toán biên. Mô hình đề xuất gồm ba lớp kiểm chứng phối hợp: kiểm tra tính toàn vẹn nội dung, xác thực nút mạng dựa trên điểm tin cậy động và xác minh tính nhất quán của hệ thống. Mỗi lớp được mô hình hóa bằng cơ chế HMAC-SHA256 kết hợp quy trình dẫn xuất khóa nhằm tăng cường bảo vệ dữ liệu. Nghiên cứu cũng tích hợp mô hình suy giảm và phục hồi độ tin cậy theo hàm mũ để thích ứng với thay đổi trạng thái an ninh. Kết quả thử nghiệm cho thấy phương pháp đề xuất chống giả mạo và mạo danh hiệu quả nhưng vẫn duy trì chi phí tính toán thấp. Đánh giá thực nghiệm chứng minh cấu trúc MAC phân cấp giúp nâng cao khả năng mở rộng, tăng độ bền trước nhiều kịch bản tấn công và hỗ trợ xác minh hiệu quả cho bộ nhớ đệm của website.

Từ khóa - Bảo mật ứng dụng web; Tính toàn vẹn bộ nhớ đệm web; Điện toán biên; Xác thực phân cấp

1. Introduction

Web caching plays a central role in accelerating modern web applications, especially as global content delivery infrastructures continue to scale in size and complexity. Content delivery networks handle billions of daily requests, making cache integrity essential for maintaining reliable and secure user experiences [1]. However, ensuring that cached content remains authentic becomes increasingly challenging in distributed edge computing environments, where cache replicas reside on numerous geographically dispersed and sometimes partially trusted nodes. Traditional validation mechanisms such as ETags and Last-Modified headers provide performance benefits but offer limited security guarantees, as they cannot detect malicious alterations nor prove the authenticity of cached responses in adversarial settings [2].

The shift toward edge computing further intensifies these limitations. Edge nodes differ in trustworthiness, capabilities, and security posture, and a single user request may traverse several nodes before retrieving cached content [3]. This distributed nature exposes systems to a range of attacks, including cache poisoning, node impersonation, and large-scale coordinated manipulation. Existing cryptographic approaches typically apply a single-level Message Authentication Code (MAC) to each cache entry [4]. While such methods provide strong per-entry integrity, they cannot distinguish between types of failures, authenticate the node responsible for generating the cache entry, or ensure consistency across the broader caching infrastructure.

To address these shortcomings, this paper presents a practical hierarchical MAC framework for web cache integrity verification in edge-computing environments. Rather than introducing a new cryptographic primitive, the framework combines three complementary verification layers: content integrity protection, node-aware authentication supported by dynamic trust scoring, and lightweight system-wide consistency checking. This layered design helps improve fault localization and operational visibility in distributed cache deployments, where security failures may arise from content tampering, compromised edge nodes, or broader inconsistencies across the caching infrastructure.

2. Related Works

Current approaches to cache integrity detection can be categorized into several distinct methodologies, each with specific strengths and limitations in addressing modern web application security requirements:

- Hash-based Validation: Simple hash-based approaches compute cryptographic hashes of cached content to detect modifications. While computationally

efficient, these methods lack authentication capabilities and cannot distinguish between legitimate updates and malicious tampering. Content hashes alone provide integrity verification but offer no protection against sophisticated attacks where attackers can replace both content and corresponding hash values [5].

- Single-level MAC Systems: Many existing systems implement single-level MAC verification using algorithms such as HMAC-SHA256 [4]. These approaches provide strong cryptographic guarantees for individual cache entries but lack the granularity needed for complex distributed environments. Single-level systems treat all integrity failures uniformly, making it difficult to diagnose the source and nature of security breaches in large-scale deployments.

- Blockchain-based Integrity: Recent studies have investigated blockchain and distributed ledger technologies for cache integrity verification [6]. These approaches offer strong consistency guarantees and tamper-evident logging capabilities. However, the consensus mechanisms required for blockchain operation introduce substantial latency and resource consumption that conflict with the performance requirements of edge computing environments.

The limitations of existing cache integrity verification techniques, combined with the evolving security requirements of modern edge computing environments, provide strong motivation for developing a hierarchical MAC-based approach that addresses current gaps while maintaining practical applicability.

Compared with conventional single-level MAC schemes, the proposed framework emphasizes layered diagnosis in addition to integrity verification. In particular, Level 1 focuses on content authenticity, Level 2 associates verification with node context and trust status, and Level 3 provides a lightweight cross-node consistency signal for operational monitoring. The purpose of this design is not to replace all existing integrity mechanisms, but to provide a practical and low-overhead solution for edge cache systems that require incremental deployment and more interpretable verification outcomes.

3. Methodology

3.1. Overview of Integrity Verification of Web Cache

Web cache integrity verification has evolved from simple validation mechanisms to sophisticated cryptographic approaches as the complexity and security requirements of modern web applications have increased. Traditional web caching systems primarily focused on performance optimization rather than security guarantees, leading to vulnerabilities in content authenticity and integrity assurance.

The transition to edge computing paradigms has further complicated cache integrity verification. Edge nodes operate with varying trust levels, network conditions, and security capabilities [7]. This heterogeneous environment requires more sophisticated integrity verification mechanisms that can adapt to different security contexts while maintaining efficient performance characteristics suitable for real-time web applications.

3.2. Motivation for Proposed Approach

Existing cache integrity solutions often fall short in addressing the practical demands of distributed web systems. Traditional MAC-based approaches provide binary security decisions without exposing the root causes of integrity failures, making it difficult for administrators to diagnose and respond to security incidents effectively. Moreover, these systems typically operate under static security assumptions that fail to account for the dynamic nature of edge computing environments, where nodes experience varying network conditions and threat levels.

A hierarchical approach to cache integrity verification offers several important advantages. Firstly, it enables granular security analysis by decomposing the integrity problem into distinct layers: content authenticity, node authentication, and system-wide consistency. This decomposition allows administrators to identify exactly where and how security breaches occur, facilitating more targeted incident response and system debugging. Secondly, a hierarchical structure naturally accommodates dynamic trust models through mathematical frameworks such as exponential decay functions, which capture how trust degrades over time, and recovery mechanisms that allow trust rehabilitation based on demonstrated reliability. Finally, the hierarchical design enables selective verification strategies where different security levels can be applied according to content criticality and node trust levels, thereby supporting a balance between strong security guarantees and operational efficiency.

These considerations motivate the development of a comprehensive hierarchical MAC construction that combines cryptographic rigor with practical deployment flexibility, enabling both fine-grained security analysis and adaptive operation in dynamic edge computing environments.

3.3. Overview of the Proposed System

Our system model focuses on typical web application caching scenarios where content must be delivered quickly and maintaining integrity guarantees simultaneously.

We consider a web cache system consisting of:

- Origin Servers: Web servers hosting the original content (HTML, CSS, JavaScript, Images);

- Edge Cache Nodes: Distributed cache servers that store copies of web content close to users;

- Web Clients: Browsers and applications that send requests for web content or services.

Each cache entry represents a web resource, identified by its URL and contains the actual content with metadata (content type, timestamp, size).

3.4. Trust Model for Web Applications

In web applications, different edge nodes may have different trust levels, which are based on:

- Geographic Location: Nodes in some regions may be more vulnerable to attacks;

- Infrastructure Provider: Different hosting providers with different security standards;

- Historical Performance: Nodes with good performance history are more trusted;

- Security Compliance: Nodes meeting security certifications receive higher trust scores.

We represent each node's trust level as a normalized score in the range [0,1], where higher values indicate greater confidence in the node's behavior and operational reliability.

3.5. How HMAC works

HMAC (Hash-based Message Authentication Code) works like a digital fingerprint, depending on both the message and a secret key.

Basic HMAC Formula:

$$HMAC(K, M) = H((K \oplus op)||H((K \oplus ip)||M)) \quad (1)$$

where:

- K is our secret key;

- M is the message we want to protect;

- H is a hash function (SHA-256 for example);

- $\oplus$ means XOR;

- $||$ means sticking two pieces of data together;

- op and ip: fixed padding values.

Without the secret key K, an attacker simply cannot forge a valid HMAC. Even if they tweak just one bit in the message M, the resulting HMAC will look completely different. This happens because hash functions have this neat property called the "Avalanche Effect", it says that tiny changes lead to dramatically different outputs.

For any attacker trying to forge an HMAC without the key, their chance is:

$$SP \leq \frac{1}{2^{256}} \quad (2)$$

where SP is the Success Probability.

3.6. Key Derivation - Making Multiple Keys from One

We are not using the same key everywhere, we create different keys for each level, so if one gets compromised, the others are still safe.

Key Derivation Formula:

$$K_{level} = HMAC(K_{master}, context_{level}) \quad (3)$$

For example, let's say our master key is "$SecKey$". We create these key levels:

- Level 1 key = $HMAC("SecKey", "context1")$;

- Level 2 key = $HMAC("SecKey", "context2")$;

- Level 3 key = $HMAC("SecKey", "context3")$.

If an attacker gets hold of one derived key (say, Level 1), they still can't work out the master key since HMAC only works one way, or figure out other level keys since each one uses its own unique context.

3.7. Trust Score Optimization

Trust Decay Function: In real cache systems, a node's trustworthiness tends to drop off naturally over time without fresh checks on its reliability. We capture this behavior using an exponential decay function:

$$T(t) = T_0 \cdot e^{-\lambda t} \quad (4)$$

where:

- $T(t)$ is the trust score at time t;

- T_0 is the initial trust score (when $t = 0$);

- λ is the decay constant (how fast trust decreases);

- e is Euler's number (approximately 2.718).

Below is a diagram of the Trust Decay Function when $\lambda = 0.001$ and the initial trust score T_0 equals to 0.95:

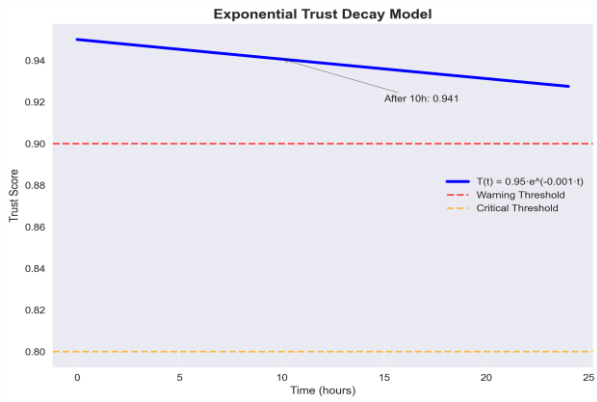

Figure 1. *Exponential Trust Decay Model*

The decay model $T(t) = 0.95 \cdot e^{-0.001 \cdot t}$ illustrates how trust levels decrease over time, dropping from a starting point of 0.95 to about 0.941 after 10 hours. In our setup, we've set key thresholds at 0.9 for warnings and 0.8 for critical alerts, which should give enough time for fixing. It takes around 54 hours to hit that warning mark, and about 223 hours before reaching the critical level.

Trust Recovery Function: When we receive positive confirmation from a node (successful operations), we can boost the trust score using:

$$T_{new} = T_{old} + (1 - T_{old}) \cdot (1 - e^{-\alpha \cdot S}) \quad (5)$$

where:

- S is the success rate of recent operations (0 to 1);

- α is the recovery rate constant;

- $1 - T_{old}$ represents how much room there is for improvement.

If T_{old} is already high (close to 1), there's little room for improvement. If $S = 1$ (perfect rate), trust increases significantly. If $S = 0$ (completely failed), trust doesn't increase. Finally, the exponential term ensures smooth, natural-looking changes

These exponential functions have some mathematical properties:

- They're smooth and continuous in the graph;

- They naturally bound values between 0 and 1;

- They're computationally efficient to calculate.

This mathematical approach ensures our cache system adapts to changing conditions while maintaining stability and predictability.

In the two functions above, λ denotes the decay constant. It determines the sensitivity of the trust value to

the passage of time in the decay model. A larger value of λ causes the trust score to decrease more rapidly, whereas a smaller value leads to a slower decline. The constant α denotes the recovery rate. It determines the sensitivity of the trust value to positive observations in the recovery model. A larger value of α results in faster trust restoration, while a smaller value yields a more gradual recovery.

In practice, λ and α should be selected according to the monitoring frequency and stability of the edge-cache environment. A larger λ is suitable for environments where trust should decrease quickly when no recent validation is available, while a smaller λ is more appropriate for stable nodes with predictable behavior. Similarly, a larger α allows faster trust recovery after successful observations, whereas a smaller α gives a more conservative recovery process. Therefore, these parameters can be tuned to balance responsiveness and stability in the trust-management process.

Now we present a diagram of the Trust Decay Function when $S = 0.95$ (can be improved) and the recovery rate constant is 2:

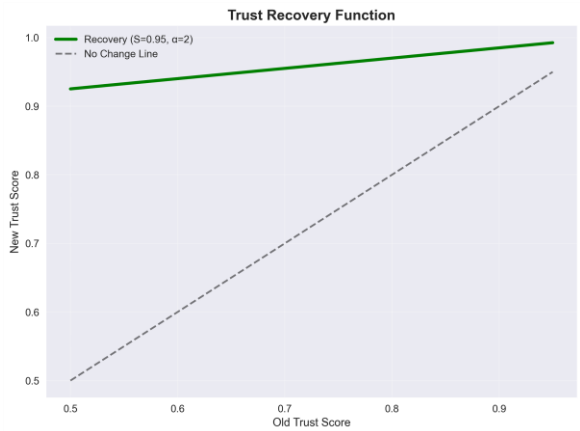

Figure 2. *Trust Recovery Function*

The recovery function, parameterized with $S = 0.95$ and $\alpha = 2$, shows reliable trust restoration, no matter the initial trust value. The green recovery curve always stays above the diagonal baseline, highlights how well the trust evaluation can be better even from a low point like 0.5.

3.8. Hierarchical MAC Construction for Web Caches

Our three-level hierarchical MAC system tackles different aspects of web cache protection, with each layer backed by mathematical guarantees.

3.8.1. Level 1: Content Integrity Verification

Level 1 focuses on verifying that the web content hasn't been modified.

$$MAC_1 = HMAC(K_1, C \,||\, L \,||\, T_s) \quad (6)$$

where C is the content, L is the URL and T_s is the timestamp when the key is generated.

Security Analysis: If an attacker tries to modify the content while keeping the same MAC, they need to find a collision. The probability for this to happen is:

$$P(\sigma) \leq \frac{1}{2^{256}} \approx 8.6 \times 10^{-78} \quad (7)$$

where, $P(\sigma)$ is the probability of Successful Tampering.

3.8.2. Level 2: Node Authentication

Level 2 verifies that the cache entry comes from a legit edge node and meets the node's trust level using our exponential trust model.

$$MAC_2 = HMAC(K_2, MAC_1||node\_id||trust\_score) \quad (8)$$

We compute the dynamic trust score by combining recent operational success, compliance status, and time-based decay into a bounded value in the range [0,1]. This score is not intended to serve as a complete security guarantee by itself; instead, it acts as an additional contextual signal that is cryptographically bound to the Level 2 MAC. As a result, an attacker would need not only to forge the MAC but also to reproduce a trust value consistent with the observed node state.

$$T_{dyn} = T(t) \times \frac{S_{opr}}{T_{total}} \times C_{factor} \quad (9)$$

where, S_{opr} stands for successful operations, T_{total} stands for total operations, C_{factor} stands for the compliance factor and $T(t)$ is calculated using our exponential decay model in the section before.

Security Analysis: If an attacker tries to impersonate a node, they would need to forge both the MAC and somehow match the dynamically changing trust score. The probability of them pulling this off is:

$$P(\varphi) \leq \frac{1}{2^{256}} + P(\text{ϑ}) \quad (10)$$

where, $P(\text{ϑ})$ is the probability of Trust Score Manipulation.

Since trust scores are also protected by MACs and change exponentially over time, that second term works out to be around $\frac{1}{2^{256}}$ as well, which keeps the overall probability extremely low.

3.8.3. Level 3: System-wide Consistency

Level 3 provides a lightweight system-wide consistency check intended to detect abnormal changes in the overall cache environment. Instead of relying only on node count and a single average trust value, the system state is derived from a compact summary that may include the number of active nodes, a coarse trust distribution, a configuration version identifier, and a synchronization epoch. This design remains lightweight while reducing dependence on a single aggregated indicator.

System State Calculation: The system state is computed as a hash of global information:

$$S_s = SHA256(N_{count} \,||\, A_{trust} \,||T_s) \quad (11)$$

where, S_s is the System State, N_{count} is Node Count, and A_{trust} is Average Trust.

For example, if we have 10 nodes with average trust 0.92 at timestamp 1693123456, then:

$$S_s = SHA256(\text{"10"}||\text{ "0.92"} \,||1693123456) \quad (12)$$

For a system-wide attack to actually work, attackers have to:

1. Take over multiple nodes at the same time.

2. Keep everything looking consistent across all the nodes they've hijacked.

3. Successfully fake all three MAC levels.

The probability of success decreases exponentially with the number of nodes:

$$P(\alpha) \leq \left(\frac{1}{2^{256}}\right)^{N_d} \quad (13)$$

where N_d is the number of nodes and $P(\alpha)$ is the probability of System-wide Attack to be successful.

With just 3 nodes, this probability drops to roughly $\frac{1}{2^{768}}$, which is astronomically tiny.

4. Implementation

Here we present models that demonstrate how our hierarchical MAC system really operates in practical, real-world setups.

4.1. Integration with Web Cache Systems

Our hierarchical MAC system can be integrated with existing web cache systems through:

HTTP Header Extensions:

X-Cache-MAC-L1: base64-encoded-level1-mac;

X-Cache-MAC-L2: base64-encoded-level2-mac;

X-Cache-MAC-L3: base64-encoded-level3-mac;

X-Cache-Node-ID: edge-node-identifier;

X-Cache-Trust: node-trust-score.

Cache Entry Structure:

Each cache entry includes:

- Original web content;
- Content hash (SHA-256);
- Timestamp and TTL;
- Three MAC values;
- Node and system metadata.

4.2. Key Management for Web Applications

The master key is generated using a cryptographically secure random number generator and stored securely, such as within a hardware security module [8]. For key distribution purposes, keys are transmitted to edge nodes through secure channels. This includes initial deployment where keys get embedded during the node setup phase, key rotation utilizing secure update protocols with proper authentication mechanisms, and emergency revocation capabilities that provide immediate key invalidation when needed.

4.3. Error Handling and Fallback Mechanisms

When MAC verification fails, the system should:

- Log the failure with detailed information
- Identify which level failed to help diagnose the issue
- Fetch fresh content from the origin server

4.4. Cache Entry Creation Process

This model shows how web content is stored in the cache with hierarchical MAC protection.

Algorithm 1: Web Cache Entry Creation

Require: Web content C, URL U, node ID N, trust score T
Ensure: Cache entry with three-level MAC protection
1:{Step 1: Initialize system}
2: $timestamp \leftarrow current\_time()$
3: $content\_hash \leftarrow SHA256(C)$
4:{Step 2: Compute Level 1 MAC (Content Integrity)}
5: $message\_1 \leftarrow C||U||timestamp$
6: $MAC_1 \leftarrow HMAC(K_1, message\_1)$
7:{Step3: Compute Level 2 MAC (Node Authentication)}
8: $message\_2 \leftarrow MAC_1||N||T$
9: $MAC_2 \leftarrow HMAC(K_2, message\_2)$
10:{Step 4: Compute Level 3 MAC (System Consistency)}
11: $system\_state \leftarrow get\_global\_state()$
12: $message\_3 \leftarrow MAC_2||system\_state$
13: $MAC_3 \leftarrow HMAC(K_3, message\_3)$
14:{Step 5: Create cache entry}
15: $entry \leftarrow CacheEntry(U, content\_hash, timestamp, MAC1, MAC2, MAC3, |C|, TTL)$
return $entry = 0$

4.5. Cache Entry Verification Process

The verification flow begins with a hash check to first ensure that the content hasn't been corrupted. This is followed by Level 1, where the content integrity is verified using the original content. In Level 2, it verifies node authentication through the relevant node information. After that, Level 3 confirms system consistency by leveraging the global state. Finally, a TTL check is performed to make sure the cache entry hasn't expired.

Algorithm 2: Web Cache Entry Verification

Require: Cache entry E, original content C, requesting node N, trust score T
Ensure: Verification result (valid/invalid) with details
1: {Step 1: Initialize verification tracking}
2: $details \leftarrow (level1: false, level2: false, level3: false)$
3: {Step 2: Verify content hash}
4: $expected\_hash \leftarrow SHA256(C)$
5: **if** $expected\_hash \neq E.content\_hash$ **then return** ($false$, "Content hash mismatch", $details$)
6: **end if**
7: {Step 3: Verify Level 1 MAC}
8: $message\_1 \leftarrow C||E.url||E.timestamp$
9: $expected\_MAC\_1 \leftarrow HMAC(K_1, message\_1)$
10: **if** $expected\_MAC\_1 \neq E.mac\_level\_1$ **then return** ($false$, "Level 1 verification failed", $details$)
11: **end if**
12: $details.level1 \leftarrow true$
13: {Step 4: Verify Level 2 MAC}
14: $message\_2 \leftarrow E.mac\_level\_1||N||T$
15: $expected\_MAC\_2 \leftarrow HMAC(K_2, message\_2)$
16: **if** $expected\_MAC\_2 \neq E.mac\_level\_2$ **then return** ($false$, "Level 2 verification failed", $details$)
17: **end if**
18: $details.level\_2 \leftarrow true$
19: {Step 5: Verify Level 3 MAC}
20: $system\_state \leftarrow get\_global\_state()$
21: $message\_3 \leftarrow E.mac\_level\_2||system\_state$
22: $expected\_MAC\_3 \leftarrow HMAC(K_3, message\_3)$
23: **if** $expected\_MAC\_3 \neq E.mac\_level\_3$ **then return**

($false$, "Level 3 verification failed", $details$)
24: **end if**
25: $details.level3 \leftarrow true$
26: {Step 6: Check expiration}
27: **if** $current\ time() > E.timestamp + E.ttl$ **then return** ($false$, "Cache entry expired", $details$)
28: **end if**
29: **return** ($true$, "All verifications passed", $details$) = 0

If any verification step fails, the process terminates immediately and reports the corresponding failure level. This design allows administrators to identify whether the problem is related to content integrity, node authentication, or system-wide consistency. As a result, the verification mechanism not only detects invalid cache entries but also provides useful diagnostic information for security monitoring and incident analysis. This layered feedback helps distinguish different types of attacks or operational failures within the cache system.

4.6. Security Analysis

We consider an adversary who may observe, replay, or modify cached web content and metadata transmitted through edge cache nodes. The adversary may also attempt node impersonation, cache poisoning, or partial compromise of selected edge nodes. However, the origin server, the master key management component, and the HMAC-SHA256 implementation are assumed to be trusted. Secret keys are assumed to remain confidential, and HMAC-SHA256 is assumed to provide standard message-authentication security against forgery under unknown keys. Under these assumptions, the proposed framework focuses on detecting unauthorized content modification, inconsistent node behavior, and abnormal system-level cache states.

4.6.1. Security Properties of Each Level

Level 1 Security:

Level 1 provides content integrity through HMAC-SHA256. The security relies on:

- Key Secrecy: Attackers cannot forge MACs without knowing K_1;

- Hash Security: SHA-256 is cryptographically secure with no known practical attacks [9];

- HMAC Security: HMAC-SHA256 provides 256-bit security against forgery attacks [10].

Level 2 Security:

Level 2 provides node authentication by binding the Level 1 MAC to specific node information:

- Node Binding: Each MAC is tied to a specific node ID and trust score;

- Trust Integration: Trust scores are cryptographically protected and dynamically updated;

- Hierarchical Linking: Level 2 depends on Level 1, so both must be valid.

Level 3 Security:

Level 3 provides system-wide consistency checking:

- Global State Protection: System state is cryptographically verified;

- Complete Chain: All three levels must be valid for successful verification;

- System-wide Attack Detection: Coordinated attacks affecting global state are detected.

4.6.2. Mathematical Proof for the Security Performance

Single-Level MAC Security:

A traditional single-leveled MAC system has security:

$$S_{single} = \frac{1}{2^{256}} \tag{14}$$

Our Hierarchical System Security:

In the proposed hierarchical system, an attacker must satisfy the verification conditions across all three levels for a malicious cache entry to be accepted:

$$S_{hierarchical} = \frac{1}{2^{256}} \times \frac{1}{2^{256}} \times \frac{1}{2^{256}} = \frac{1}{2^{768}} \tag{15}$$

Security Improvement Factor:

$$d = \frac{S_{hierarchical}}{S_{single}} = \frac{1}{2^{512}} \tag{16}$$

$$I_{factor} = \frac{1}{d} = 2^{512} \tag{17}$$

where d is the Decreased Attack Chance and I_{factor} is the Security Improvement Factor.

4.6.3. Practical Security Analysis

In the first attack, content tampering, any change to cached web content will cause the Level 1 verification to fail. Since the content itself has been altered, the system immediately flags the tampering attempt, and the specific failure at this level points straight to a breach of content integrity.

In the second scenario, node compromise, if an attacker gets control of an edge node, they might be able to create valid Level 1 MACs. However, the Level 2 verification will fail once the node's trust score changes, which allows system administrators to know which node has been compromised.

Finally, in a system-wide attack scenario, where the attackers need to coordinate their actions across multiple nodes, the Level 1 and Level 2 checks could very well pass for those individual entries. However, the Level 3 verification step, which examines the overall global state, would likely fail due to those inconsistencies, allowing the system to detect the broader pattern of the attack.

4.7. Performance Analysis for Web Applications

We analyze the performance characteristics of our system in typical web application scenarios.

4.7.1. Computational Overhead

MAC Generation Cost:

For each cache entry, we compute three HMAC operations:

- Level 1: HMAC over content + URL + timestamp;

- Level 2: HMAC over Level 1 MAC + node info;

- Level 3: HMAC over Level 2 MAC + system state.

Time Complexity Analysis:

$$T_{total} = T_{lv1} + T_{lv2} + T_{lv3} \tag{18}$$

where:

- $T_{lv1} = O(content\ size)$: depends on content length;

- $T_{lv2} = O(1)$: constant time (MAC + node info is small);

- $T_{lv3} = O(1)$: constant time (MAC + system state is small).

So total time is $O(content\ size)$, equal to the total time of single-level systems.

Performance Comparison:

Table 1. *Performance Comparison for Web Cache Operations.*

Operation	Single-Level MAC	Hierarchical MAC
Generation	1 HMAC	3 HMACs
Verification	1 HMAC	1-3 HMACs (selective
Storage Overhead	32 bytes	96 bytes
Network Overhead	32 bytes	96 bytes

The hierarchical MAC framework requires three HMAC computations during generation compared to one for single-level MAC, providing coverage across three verification layers. Verification remains adaptive, requiring one to three HMACs depending on threat conditions. Both approaches incur 32 bytes storage overhead per entry, though hierarchical MAC increases network overhead to 96 bytes due to additional trust score and consistency metadata.

4.7.2. Real-World Performance Estimates

Typical Web Content Sizes:

- HTML pages: 10-100 KB;

- CSS files: 5-50 KB;

- JavaScript files: 20-200 KB;

- Images: 50-500 KB.

HMAC-SHA256 Performance: Modern processors can compute HMAC-SHA256 at approximately 500 MB/s. For typical web content:

- 10 KB HTML page: ~ 0.02 ms per HMAC;

- 50 KB CSS file: ~ 0.1 ms per HMAC;

- 200 KB JavaScript: ~ 0.4 ms per HMAC.

Total Overhead: For a normal web page with multiple resources, the total MAC computation time is less than 5 ms, which is negligible compared to network latency (typically 50-200 ms).

5. Results and Discussion

To validate the proposed hierarchical MAC system, we implemented the framework in Python and examined its core functionalities and performance characteristics under controlled web application scenarios. The evaluation was conducted using representative test data to demonstrate the practical capabilities of the system.

5.1. Experimental Setup

Implementation Details:

- Language: Python 3.12;

- Hash Function: SHA-256;

- Key Size: 256 bits, for all derived keys;

- Test Environment: Web server with multiple edge node simulation.

5.2. Test Content

We tested with typical web content:

- HTML pages: Various sizes, from 5KB to 100KB;

- CSS stylesheets: 10KB to 50KB;

- JavaScript files: 20KB to 200KB;

- Static resources: Images and media files;

- Edge nodes: Sample node information (excerpt from a larger dataset due to its extensive size).

Table 2. *Example JSON Log Data (Truncated for Brevity)*

```
[
        {
        'node_id': 'cdn-us-west-2-server-04',
        'location': 'Oregon, USA',
        'trust_score': 0.96,
        'capacity_gbps': 90,
        'uptime_percentage': 99.8
        },
        {
        'node_id': 'cdn-ap-northeast-1-server-05',
        'location': 'Tokyo, Japan',
        'trust_score': 0.94,
        'capacity_gbps': 70,
        'uptime_percentage': 99.6
        }
]
```

5.3. Performance Results

MAC Generation Times:

Table 3.*MAC Generation Performance.*

Content Size	Level 1	Level 2	Level 3
5 KB	0.12 ms	0.05 ms	0.05 ms
25 KB	0.35 ms	0.05 ms	0.05 ms
100 KB	1.2 ms	0.05 ms	0.05 ms

The hierarchical MAC framework achieves efficient performance across different content sizes. MAC generation times are 0.12 ms at Level 1 for 5 KB content, while Levels 2 and 3 consistently maintain 0.05 ms regardless of content size. Even with 100 KB blocks, the overhead remains manageable at 1.2 ms for Level 1, demonstrating that the multi-layer approach effectively prevents performance bottlenecks in edge environments.

Scalability Consideration:

The evaluation results indicate that the additional overhead of the proposed framework mainly comes from a small number of HMAC computations per cache entry. Since *Level 2* and *Level 3* operate on compact metadata rather than full content, their processing time remains almost constant across different content sizes in the current experiment. This suggests that the framework can scale reasonably with increasing cache entries and edge nodes, provided that metadata synchronization and key management are efficiently maintained. A larger scale deployment evaluation with more edge nodes is left for future work.

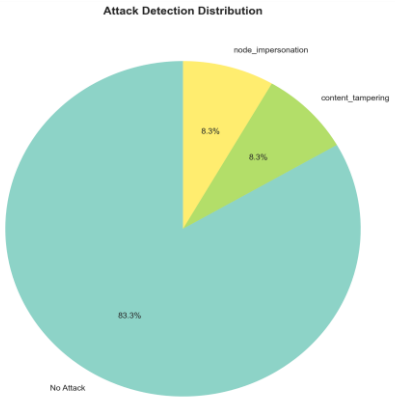

Figure 3. *Attack Detection Distribution*

The pie chart shows there's an imbalance, 83.3% of the traffic being benign stuff, and attacks make up just 16.7%. Those attack cases are pretty evenly divided between node impersonation and content tampering, each sitting at around 8.3%. This kind of distribution actually makes sense when in real world networks, most of the traffic flowing through is legitimate.

5.4. Comparision with other Tools

Table 4. *Security Feature Comparison*

Feature	ETag Only	Single MAC	Hierarchical MAC
Content Integrity	Weak	Strong	Strong
Node Authentication	None	None	Strong
System Consistency	None	None	Strong
Attack Type Detection	None	Limited	Strong
Trust Integration	None	None	Yes

The comparison suggests that the hierarchical MAC framework provides broader verification coverage than ETag Only and Single MAC approaches in the prototype setting considered in this paper. Its main advantage lies in combining content verification, node-aware authentication, and a lightweight system-level consistency signal within a unified workflow. These results should be interpreted as a preliminary functional comparison rather than a comprehensive benchmark against all modern integrity verification architectures.

6. Conclusion

This paper presents a hierarchical MAC framework for verifying the integrity of web caches in edge-computing environments. By combining content integrity verification, node-aware authentication, and system-wide consistency checking, the proposed approach provides a structured and practical security mechanism for distributed cache systems. The study shows that the hierarchical design improves the interpretability of verification outcomes while maintaining low computational overhead for typical web workloads. These results indicate that the framework is a feasible and effective solution for strengthening cache integrity in modern edge infrastructures. Future work may extend this direction through broader comparative evaluation, more detailed adversarial analysis, and deployment studies in larger-scale operational settings.

REFERENCES

[1] A. Vakali and G. Pallis, "Content delivery networks: Status and trends", *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, 2003. https://doi.org/10.1109/MIC.2003.1250586

[2] *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, Internet Engineering Task Force, June 1999.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges", *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016. https://doi.org/10.1109/JIOT.2016.2579198

[4] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication", in *Advances in Cryptology – CRYPTO '96*, Santa Barbara, CA, USA, August 18–22, 1996, Lecture Notes in Computer Science, vol. 1109, Springer, 1996, pp. 1–15.

[5] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance", in *Fast Software Encryption (FSE 2004)*, Lecture Notes in Computer Science, vol. 3017, Springer, 2004, pp. 1–30.

[6] A. Alhussen and E. Arslan, "Blockchain-based integrity verification for file transfers", in *2020 IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA, 2020, pp. 3255–3264.

[7] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing Cache Data Integrity in the Edge Computing Environment", *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1210–1223, 2021. https://doi.org/10.1109/TPDS.2020.3043755

[8] *Recommendation for Key Management: Part 1 – General*, NIST Spec. Publ. 800-57 Pt. 1 Rev. 5, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2020.

[9] *Secure Hash Standard (SHS)*, FIPS PUB 180-4, National Institute of Standards and Technology, Gaithersburg, MD, USA, August 2015.

[10] M. Bellare, "New proofs for NMAC and HMAC: Security without collision-resistance", in *Advances in Cryptology – CRYPTO 2006*, Santa Barbara, CA, USA, Aug. 20–24, 2006, Lecture Notes in Computer Science, vol. 4117, Springer, 2006, pp. 602–619.