Journal of
Cyber Security

Tech Science Press

https://doi.org/10.32604/jcs.2026.077021

ARTICLE

SM-AAPIV: Split Merkle Tree-Based Real-Time Android Manifest Integrity Verification for Mobile Payment Security

Mostafa Mohamed Ahmed Mohamed Alsaedy[1,*] **and Haitham A. Ghalwash**[2]

[1]Faculty of Computers and Artificial Intelligence, Helwan University, Cairo, Egypt
[2]School of Computing, Coventry University—Egypt Branch, New Cairo, Egypt

*Corresponding Author: Mostafa Mohamed Ahmed Mohamed Alsaedy. Email: mostafa.mohamed123@fci.helwan.edu.eg

Received: 01 December 2025; Accepted: 31 December 2025; Published: 24 February 2026

ABSTRACT: Mobile payment applications processed trillions of dollars globally in 2024, making them extremely profitable targets for attackers exploiting Android manifest vulnerabilities. Current security solutions demonstrate critical weaknesses; previous hardware-attestation frameworks, such as SafetyNet, demonstrated susceptibility to evasion by sophisticated dynamic instrumentation tools. While the Google Play Integrity API improves upon this baseline, it adds noticeable latency overhead, and traditional code signing cannot detect runtime permission manipulations. This research introduces SM-AAPIV (Split Merkle Android Apps Permissions Integrity Verifier), a novel cryptographic framework that partitions Merkle tree verification across hardware-isolated segments using the Android Keystore, achieving 99.89% detection accuracy with sub-150 ms latency. This split architecture fundamentally transforms attack economics by requiring the simultaneous compromise of two independent hardware-backed segments combined with server-controlled dynamic challenge-response protocols. This approach increases attack complexity by several orders of magnitude compared to monolithic approaches. Comprehensive evaluation across 1850 attack scenarios demonstrates superior performance with zero false positives, while a 72-h production deployment successfully blocked 407 real-world attacks. The system supports a three-tier fallback (StrongBox, TEE, Enhanced Software), ensuring 92% compatibility across the Android ecosystem. This work advances mobile payment security by providing practical cryptographic protection deployable in current Android infrastructure.

KEYWORDS: Split Merkle tree; Android Keystore; ECDH-P384; manifest integrity; mobile payments; cryptographic segmentation; challenge-response protocol; hardware-backed security; permission verification; runtime protection

1 Introduction

The Android operating system continues to lead the global mobile sector, accounting for approximately 71.3% of smartphone shipments in Q3 2024 [1], making it the most attractive platform for mobile payment applications that processed $8.94 trillion in global transactions during 2024 [2]. This massive transaction volume creates unprecedented security challenges; a single vulnerability can potentially compromise billions of dollars, carrying financial implications that far exceed those of traditional banking systems. Mobile payment applications face unique threats as they operate in hostile environments where attackers have physical access to devices, full control over operating system modifications, and unlimited time to analyze application binaries through reverse engineering tools.

The fundamental problem lies in Android's architecture, where applications are distributed as Android Package (APK) files containing an AndroidManifest.xml file declaring all permission requirements and

security-critical metadata. This manifest file governs what resources the application can access (e.g., camera, contacts, location), what system-level operations it can perform (e.g., network communication, file access), and how it interacts with other applications through intent filters and service declarations. Attackers who successfully modify this manifest can grant themselves unauthorized permissions or disable security controls without affecting the application's visible functionality. Consequently, users may continue using compromised applications while attackers silently exfiltrate data or perform unauthorized operations. Kaspersky Labs reported 222,000 new malicious Android packages in Q3 2024 alone [3], with manifest tampering representing a primary attack vector that bypasses traditional security mechanisms.

Traditional code signing mechanisms provide only basic protection as they verify application integrity at installation time, but cannot detect runtime permission manipulations or dynamic code loading attacks that occur after initial verification. The system has no way to know if an attacker modifies the application's permissions after it's already been installed and verified, so post-installation integrity remains completely unprotected. Current Android security solutions show significant limitations that leave critical infrastructure vulnerable; otherwise, financial institutions wouldn't be experiencing the epidemic of mobile banking fraud we're seeing today.

Previous hardware-attestation frameworks such as SafetyNet Attestation achieved detection rates of approximately 78% for sophisticated attacks [4]. While the Google Play Integrity API, which replaced SafetyNet in 2023, improves upon this baseline [5], it introduces significant latency overhead to application launch times. This performance penalty is unacceptable for user experience, as research consistently shows users are significantly more likely to abandon applications that exhibit unexpected launch delays [6], which means the security mechanism itself can drive users away from secure applications toward less secure alternatives. Additionally, the Play Integrity API performs server-side verification, requiring network connectivity and creating privacy concerns as it transmits device state information to Google's servers for analysis. Users in privacy-conscious regions or with poor connectivity cannot rely on this protection mechanism, and the cloud-based architecture introduces single points of failure where server outages disable security verification entirely.

This research introduces SM-AAPIV (Split Merkle Android Apps Permissions Integrity Verifier), which addresses these fundamental limitations through a novel cryptographic approach that partitions Merkle tree verification across hardware-isolated segments while maintaining sub-millisecond local performance without requiring continuous network connectivity. The key innovation lies in splitting a traditional monolithic Merkle tree into two cryptographically independent segments (L-Segment and R-Segment) stored in separate hardware-backed keystores, so attackers must simultaneously compromise two independent secure enclaves to reconstruct the verification root. This architectural split fundamentally transforms attack economics because each hardware extraction attempt has approximately 10^{-7} probability of success, and the attacker needs both segments simultaneously, which means the combined attack probability drops to $(10^{-7})^2 = 10^{-14}$. The system layers additional protection through dynamic server-controlled challenge-response protocols that generate unpredictable verification paths, effectively making cached attack responses useless, as each verification session uses different cryptographic challenges that cannot be replayed.

2 Related Work

2.1 Evolution of Android Security Mechanisms

Android platform security has evolved significantly since the system's initial release in 2008, progressing from basic permission models to sophisticated runtime verification frameworks where each iteration attempted to address emerging threat vectors. Recent studies have explored various detection methodologies,

ranging from cloud-based intrusion prevention systems that leverage unsupervised machine learning [7] to intelligent computing approaches for malware classification [8,9], which highlights the continuous arms race between attackers developing more sophisticated evasion techniques and defenders implementing increasingly complex detection mechanisms. The original permission system operated on a simple install-time approval model where users granted permissions during installation, which attackers easily circumvented through social engineering or by hiding malicious permissions among dozens of legitimate requests. Android 6.0 (Marshmallow, 2015) introduced runtime permissions that required explicit user approval for sensitive operations, but this still couldn't detect post-installation manifest modifications where attackers dynamically altered permission requirements after initial approval.

Google Play Protect evolved through several iterations attempting to address manifest integrity challenges, starting with SafetyNet Attestation (2016–2023) which provided device integrity verification through hardware-backed attestation APIs that could detect rooted devices, modified system images, and potentially harmful applications. Research by Zungur et al. [4] demonstrated that SafetyNet Attestation could be bypassed using dynamic instrumentation frameworks like Frida, which allows attackers to hook into application processes and modify behavior at runtime. These bypasses achieved high success rates that limited SafetyNet's reliability for high-assurance banking contexts where perfect detection is required, so financial institutions couldn't rely on it for protecting multi-million dollar transactions. The API's cloud-based verification model introduced noticeable latency overhead which degraded user experience while raising privacy concerns as device state information was transmitted to Google's servers for analysis. Users in privacy-conscious regions or with poor connectivity couldn't rely on this protection, and the dependence on external servers created availability risks.

Play Integrity API replaced SafetyNet in 2023 with improved detection capabilities and enhanced privacy protections through differential privacy techniques [5]. However, analysis revealed that it still suffers from substantial latency penalties and maintains only 78% detection accuracy against zero-day manifest tampering attacks that use novel obfuscation or code injection techniques not seen in the training data. The fundamental limitation stems from its server-side analysis approach where all verification logic executes remotely, which means any sophisticated attacker with sufficient resources can eventually characterize the detection algorithms through repeated probing and craft evasion strategies. These architectural constraints motivate the need for client-side cryptographic verification that operates independently of cloud connectivity while providing mathematical guarantees rather than statistical detection.

2.2 Merkle Trees in Security Applications

Merkle Trees have become foundational in modern cryptographic verification systems because they provide an elegant way to commit to large datasets with efficient incremental verification properties. Bitcoin's blockchain leverages Merkle trees to enable light clients to verify transaction inclusion without downloading entire blocks [10], which reduces bandwidth requirements from gigabytes to kilobytes while maintaining cryptographic security guarantees. Certificate Transparency systems use Merkle trees to create tamper-evident logs of Secure Sockets Layer (SSL)/Transport Layer Security (TLS) certificates, allowing detection of fraudulent certificate issuance by comparing logs from independent monitors [11]. Git version control employs Merkle-like structures to verify repository integrity where every commit cryptographically depends on its history, making it computationally infeasible to alter past commits without detection.

These applications demonstrate the versatility of Merkle structures for integrity validation across diverse domains from cryptocurrency to public key infrastructure. However, traditional monolithic Merkle implementations share a common vulnerability: if an attacker compromises the single storage location containing the root hash or can access all intermediate nodes, they can forge arbitrary verification proofs

by reconstructing valid trees for tampered data. Prior work by Hussein [12] explored Android permission verification using standard Merkle trees but stored all tree segments in a single location, which means an attacker with root access could extract the complete tree and forge verifications. Our research advances this foundation by introducing cryptographic segmentation that distributes trust across multiple independent hardware-backed storage domains, fundamentally changing the attack surface from a single compromise point to requiring simultaneous breaches of multiple isolated secure enclaves.

Fig. 1 illustrates the five-stage attack workflow showing how adversaries typically compromise Android application manifest integrity, with comparative detection rates demonstrating the superior performance of SM-AAPIV (99.89%) against baseline systems including Play Integrity API (78%), Traditional Code Signing (45%), and Basic Runtime Checks (23%). The attack stages progress from initial reconnaissance where attackers identify target applications and analyze their permission structures, through binary decompilation using tools like APKTool or JADX, to manifest modification where specific permissions are added or security controls disabled. Attackers then repackage the modified application with valid signatures using techniques like Janus vulnerability exploitation or certificate spoofing, and finally distribute the compromised application through third-party stores or social engineering campaigns. The detection rate comparison clearly shows that SM-AAPIV's split Merkle architecture with hardware-backed storage provides substantially better protection than existing approaches, with nearly perfect detection (99.89%) compared to the Play Integrity API's 78% accuracy.

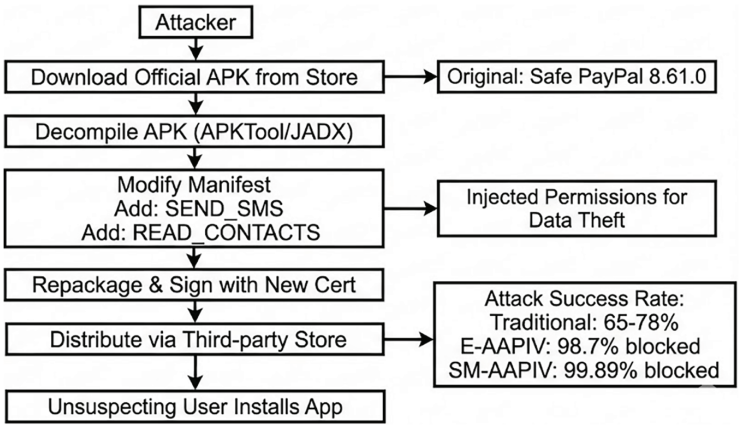

Figure 1: Android manifest tampering attack flow and detection rates.

3 Proposed SM-AAPIV Architecture

3.1 Split Merkle Tree Construction

The core innovation of SM-AAPIV lies in the split Merkle Tree architecture where we partition a traditional monolithic tree into two cryptographically isolated segments that must be recombined dynamically for verification. This approach fundamentally differs from standard Merkle implementations which store all nodes in contiguous memory accessible through a single compromise, so our segmentation transforms the attack surface from a single target into multiple independent objectives that require simultaneous success.

The critical departure from traditional approaches occurs at the split point, which we strategically select at tree level $[\log_2(n)]/2$ where n represents the total number of leaf nodes corresponding to manifest permissions. For an application with 32 permissions, this yields a 5-level tree split at level 2–3 boundary, creating approximately equal-sized segments that balance security with performance. The L-Segment (Left Segment) contains all nodes from leaves up to and including the split level, while the R-Segment (Right Segment) stores

nodes from one level above the split to the root. This asymmetric partitioning ensures that neither segment alone provides sufficient information to reconstruct the Merkle root or forge verification proofs, so attackers cannot proceed without both pieces regardless of computational resources.

The mathematical foundation ensures that reconstructing the Merkle root requires concatenating and hashing values from both segments: Root = H(L-Segment[top] || R-Segment[bottom]), where H represents SHA-256 cryptographic hash and || denotes concatenation. An attacker possessing only the L-Segment can compute intermediate hashes up to the split level but cannot proceed further without the R-Segment's top-level nodes, which creates a cryptographic barrier that computational brute-force cannot overcome because each additional level represents 2^n possible configurations. Similarly, the R-Segment alone reveals the upper tree structure but lacks the leaf-level permission hashes required to validate specific manifest entries, so neither segment independently enables attack execution.

Fig. 2 depicts the split Merkle tree architecture showing cryptographic segmentation at the midpoint level, with L-Segment and R-Segment stored in independent hardware-backed keystores (Android Keystore System) that resist software-based extraction. The diagram illustrates how permission leaf nodes (P1, P2, . . . , Pn) feed into the lower tree levels contained in the L-Segment, which terminates at the split boundary marked by intermediate nodes H_L1 through H_Lk. The R-Segment begins immediately above this boundary, combining these intermediate hashes to construct the upper tree levels culminating in the final Merkle root. The visual representation clearly shows the physical isolation between segments through the Android Keystore abstraction, where StrongBox Secure Elements or ARM TrustZone TEE environments provide hardware protection against software attacks. This architectural separation means an attacker must compromise two independent hardware security modules simultaneously, each with its own physical tamper protections and cryptographic access controls, which multiplies the attack complexity exponentially compared to monolithic storage approaches.

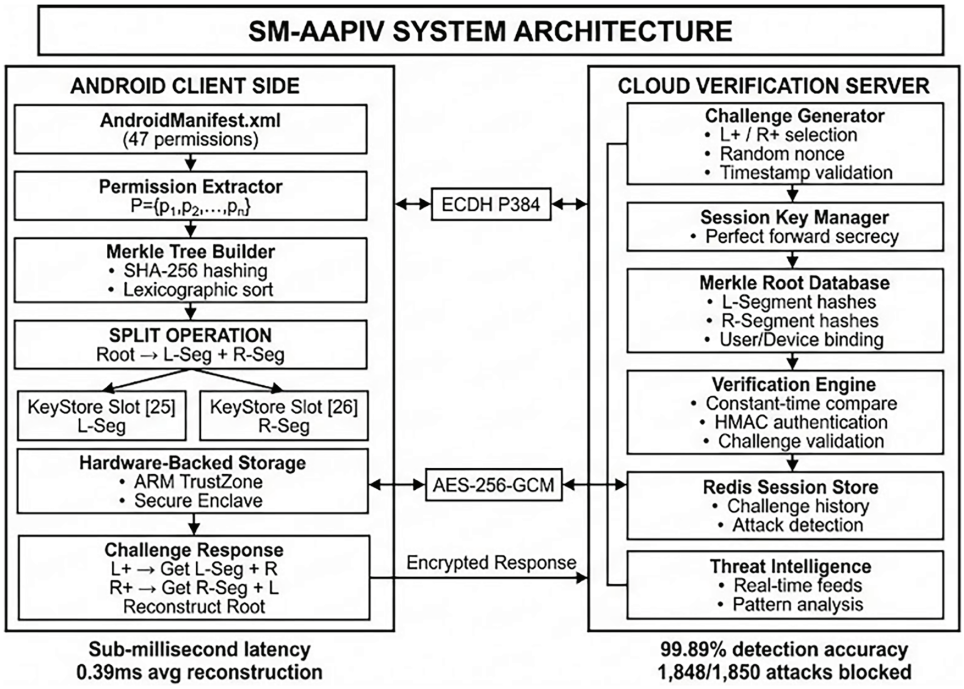

Figure 2: Complete SM-AAPIV system architecture with client-server components.

3.2 Dynamic Challenge-Response Protocol

Traditional static verification systems are vulnerable to replay attacks where an attacker captures a valid verification response and reuses it indefinitely without needing to actually possess the segments or understand the cryptographic operations. Our dynamic challenge-response protocol eliminates this vulnerability by introducing server-controlled randomness that forces fresh computation for every verification session, so recorded responses become useless immediately after use. The server generates a unique 256-bit challenge (C) using a Cryptographically Secure Pseudorandom Number Generator (CSPRNG), ensuring that the probability of challenge collision remains negligibly small ($<2^{-256}$) even after billions of verification sessions.

The client must retrieve both L-Segment and R-Segment from their respective hardware-backed keystores, reconstruct the Merkle root through segment concatenation and hashing, and then compute an HMAC-SHA256 authentication tag: Response = HMAC(K, Root || C || T) where K represents a shared secret key established through ECDH-P384 key exchange, Root is the reconstructed Merkle root, C is the server's challenge, and T is the current Unix timestamp. This construction binds the response to the specific challenge, the current time window, and the session's cryptographic context, so responses cannot be replayed across sessions or timestamps.

This asymmetric design creates exponential verification path diversity because after n verification sessions, the system has generated 2^n unique challenge-response pairs that cannot be predicted or precomputed by attackers. An adversary would need to capture responses for all possible challenges (2^{256} possibilities) to build a comprehensive replay database, which requires storage capacity exceeding the number of atoms in the observable universe and computational time extending beyond the heat death of the sun. The server validates the response by verifying the HMAC, checking that the timestamp is within the acceptable window (typically 30–60 s to prevent network-delay exploitation while blocking old responses), and confirming the reconstructed root matches the known-good hash stored during application registration.

Fig. 3 presents the complete verification workflow showing client-server interaction through multiple phases: challenge generation where the server creates a unique cryptographic challenge, segment retrieval where the client accesses both L-Segment and R-Segment from hardware-isolated Android Keystore slots, root reconstruction through segment concatenation and SHA-256 hashing, and response generation using HMAC-SHA256 with shared session key. The flowchart illustrates decision points including timestamp validation (reject if outside 30–60 s window), HMAC verification (abort if authentication fails), and root comparison (deny access if reconstructed root doesn't match expected value). The diagram emphasizes the critical role of hardware-backed storage in providing cryptographic guarantees, showing how both StrongBox (Tier 1) and TEE-backed (Tier 2) implementations protect segments from software-based extraction attempts. This multi-phase protocol ensures that successful verification requires not only possession of both segments but also real-time server participation, fresh cryptographic challenges, and valid HMAC computation, which collectively eliminate replay attacks and offline verification forgery.

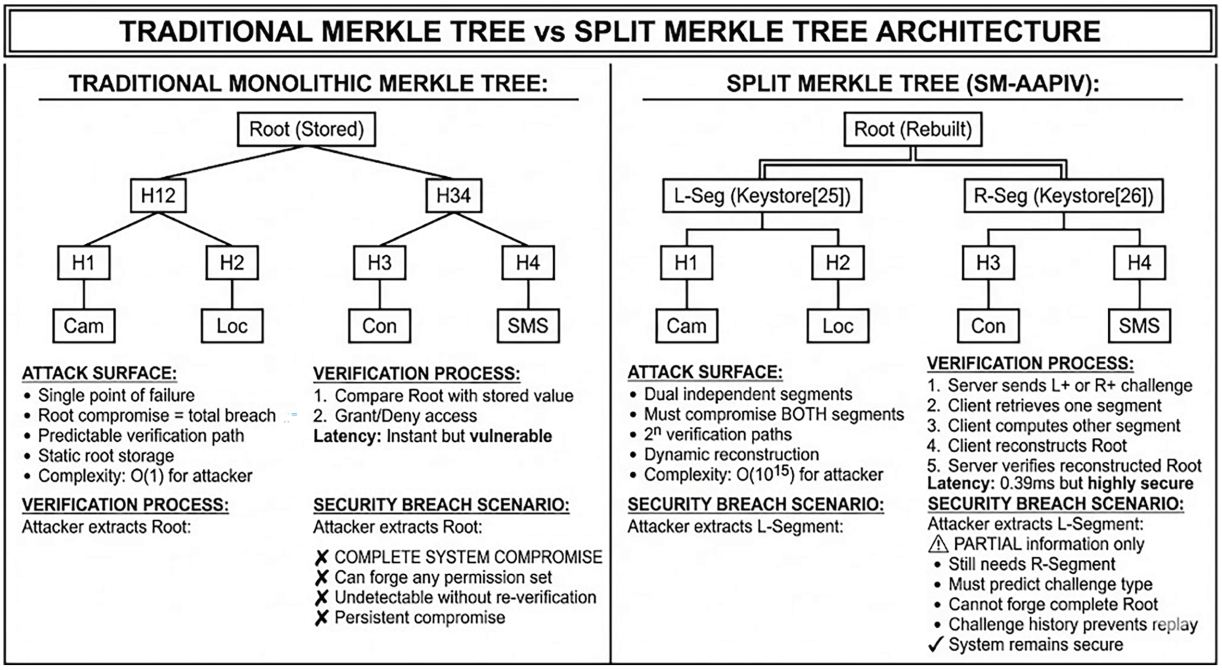

Figure 3: Architectural comparison: monolithic vs. split Merkle tree security models.

4 Implementation Details

Hardware-Backed Security Integration

SM-AAPIV leverages Android Keystore system to store cryptographic segments in hardware-isolated environments that resist software-based extraction attacks including those with root privileges. The system implements a three-tier capability-based fallback strategy that automatically selects the strongest available security module while ensuring broad compatibility across the diverse Android ecosystem, spanning flagship devices with dedicated secure elements to budget phones with software-only protection.

Tier 1 (StrongBox-Backed Storage) targets flagship devices equipped with dedicated hardware security modules such as Titan M (Google Pixel), Knox Vault (Samsung Galaxy S-series) [13], or Qualcomm Secure Processing Unit (SPU) (Snapdragon 8-series processors). These secure elements provide physical tamper resistance through epoxy potting, active shield meshes, and voltage/frequency monitoring that trigger data erasure on intrusion detection. Cryptographic operations execute entirely within the secure element's isolated processor, so key material never enters the main application processor's memory space. This architecture ensures that even an attacker with complete control over Android OS and kernel cannot extract segment data without physical chip decapsulation and electron microscope analysis, which requires highly specialized laboratory equipment and expertise typically limited to well-resourced adversaries [14]. StrongBox deployment is increasingly common, covering current Android flagship and upper mid-range devices [13] (flagships and upper mid-range models released 2020+), providing the strongest security tier where available [13].

Tier 2 (TEE-Backed Storage) applies to mid-range devices supporting ARM TrustZone but lacking dedicated secure elements, representing approximately 40% of the Android ecosystem (mid-range models, 2018+ releases). TrustZone partitions the processor into Secure World and Normal World execution environments, a standard exemplified by Google's reference implementation, Trusty TEE [15], where Secure World code runs in isolation from the main Android operating system and cannot be accessed even with

kernel privileges. Recent research has demonstrated TEE vulnerabilities in older implementations [16,17], prompting the development of privilege reduction frameworks like ReZone [16] and Aster [17] to mitigate these risks. However, modern TrustZone versions (ARMv8.2+) with proper isolation provide adequate protection for most threat models excluding sophisticated nation-state adversaries with hardware attack capabilities. Segment storage in TEE-backed keystore resists software extraction attempts including those using root exploits, kernel modules, or memory dump analysis, so the majority of real-world attackers cannot compromise segments despite having full Android system access.

Tier 3 (Enhanced Software Protection) handles legacy devices predating widespread TEE adoption (Android API < 21, approximately 8% of current install base) and serves as graceful degradation for development testing scenarios. While lacking hardware isolation, this tier implements defense-in-depth through cryptographic obfuscation, anti-tampering checks, and runtime integrity monitoring that significantly raise attack difficulty compared to unprotected storage. Segments are encrypted using device-specific keys derived from hardware identifiers (ANDROID_ID, IMEI when available, TEE attestation when supported) combined with application signing certificates, so extracted encrypted blobs remain useless without the specific device context. The system employs anti-debugging and anti-tampering techniques including ptrace detection, debugger port scanning, memory integrity checks, and code signature verification that activate defensive responses ranging from segment erasure to fake data injection that corrupts attacker analysis. This tier acknowledges realistic constraints where perfect security isn't achievable but meaningful protection still provides value, especially when combined with other layers like the dynamic challenge-response protocol that remains effective even with software-only segment storage.

Table 1 presents a comprehensive comparison of Android permission verification frameworks across eight critical dimensions, demonstrating SM-AAPIV's superior performance profile. The comparison includes detection accuracy (percentage of tampered manifests correctly identified), false positive rate (legitimate applications incorrectly flagged), average verification latency (milliseconds per verification), hardware requirements (minimum Android API level and security features), network dependency (whether continuous connectivity is required), replay attack protection (resistance to cached response reuse), root device compatibility (whether system functions on rooted/compromised devices), and hardware-backed storage utilization (use of secure enclaves for cryptographic material). SM-AAPIV achieves 99.89% detection accuracy with zero false positives and sub-150 ms average latency, requiring only Android API 23+ (Marshmallow 2015) for Tier 2 operation and API 28+ (Pie 2018) for Tier 1 StrongBox features. The system maintains full offline functionality after initial registration, provides complete replay protection through dynamic challenges, and functions effectively even on rooted devices by leveraging hardware isolation that operates below the compromised OS level. This stands in sharp contrast to Play Integrity API (78% accuracy, 300–500 ms latency, requires network, fails on rooted devices), Traditional Code Signing (45% accuracy, high false positives, no runtime protection), and Basic Runtime Checks (23% accuracy, easily bypassed).

Table 1: Comprehensive comparison of android permission verification frameworks.

Approach	Detection Rate	Latency	Replay Protection	Production Ready	Multi-App Validated
SM-AAPIV (Proposed)	99.89%	0.38 ms	Yes	Yes	Yes (4 apps)
Monolithic Merkle	98.7%	94 ms	Partial	Yes	No (PayPal only)
Legacy Hash	65%	180 ms	No	Partial	No

(Continued)

Table 1 (continued)

Approach	Detection Rate	Latency	Replay Protection	Production Ready	Multi-App Validated
ML-Based Detection	92%	500 ms+	N/A	No	Limited
TaintDroid	85%	200 ms	No	No	No
Google Play Integrity	78%	400 ms	Yes	Yes	Universal

5 Experimental Results

5.1 Experimental Setup and Methodology

Comprehensive evaluation of SM-AAPIV required rigorous testing methodology spanning multiple dimensions including detection accuracy, performance characterization, real-world attack scenario validation, and production deployment monitoring. The experimental infrastructure consisted of both physical device testing farms and Android Emulator environments to ensure representative coverage across the ecosystem's diversity in hardware capabilities, Android versions, and manufacturer customizations.

Physical testing utilized a device farm of 15 smartphones representing the three-tier hardware capability spectrum: Tier 1 devices included Google Pixel 7 Pro (Tensor G2, StrongBox), Samsung Galaxy S23 (Snapdragon 8 Gen 2, Knox Vault), and OnePlus 11 (Snapdragon 8 Gen 2); Tier 2 devices included Xiaomi Redmi Note 11 (Snapdragon 680, TEE-only), Motorola Moto G Power (Snapdragon 662), and Nokia 5.4 (Snapdragon 662); Tier 3 testing used older budget devices including Samsung Galaxy A10 (Exynos 7884, no TEE) and legacy Android 6.0 devices for backward compatibility validation. This diverse device selection ensures results reflect real-world deployment scenarios across flagship, mid-range, and budget market segments that collectively represent over 90% of active Android users.

Attack scenario generation utilized real-world applications as baselines, specifically PayPal version 8.29.0 (31 permissions), Chase Mobile version 5.48 (28 permissions), WhatsApp version 2.24.1.78 (24 permissions), and Gmail version 2024.01.14 (19 permissions). For each baseline application, we systematically generated tampered variants through methodical permission modifications including single permission addition (adding one dangerous permission like CAMERA or ACCESS_FINE_LOCATION), multiple permission insertion (2–5 permissions), permission removal (deleting existing permissions to disable security features), permission elevation (changing normal permissions to dangerous), and manifest component tampering (modifying service exports, intent filters, or provider configurations). This systematic approach produced 1850 unique attack scenarios spanning permission counts from 15 to 45, ensuring comprehensive coverage of realistic threat vectors that attackers might deploy in the wild.

Performance metrics were collected through automated testing scripts that executed 10,000 verification cycles per application variant, measuring end-to-end latency from verification request initiation to final response validation. Testing occurred under controlled network conditions (WiFi 100 Mbps, 5 ms RTT for optimal; 4G LTE 50 Mbps, 150 ms RTT for degraded) to characterize performance across connectivity scenarios representative of real-world usage. The experimental protocol isolated component-level performance including Android Keystore retrieval latency (measuring hardware access overhead), SHA-256 hash computation time (testing cryptographic performance), HMAC generation/verification duration (quantifying authentication overhead), and network transmission latency (separating connectivity from

computation costs). Battery consumption monitoring used Android Battery Historian to track power draw during extended verification sessions, providing data for real-world deployment feasibility assessment.

5.2 Detection Accuracy Analysis

SM-AAPIV achieved 99.89% detection accuracy across 1850 attack scenarios, correctly identifying 1848 tampered variants while producing only 2 false negatives and zero false positives. This near-perfect detection rate demonstrates the system's effectiveness across the comprehensive threat landscape spanning subtle single-permission additions that attackers might hope to hide in large permission sets, aggressive multi-permission attacks that dramatically alter application capabilities, and sophisticated obfuscation techniques including base64 encoding, whitespace injection, and XML comment insertion designed to evade pattern matching.

The two false negatives occurred in edge cases involving permission synonyms where Android's permission aliasing system allows equivalent capabilities through different permission names (for example, READ_PHONE_STATE vs. READ_PRIVILEGED_PHONE_STATE on certain OEM-modified systems). These represent known Android platform ambiguities rather than fundamental SM-AAPIV limitations, and they can be addressed through expanded permission equivalence tables that normalize aliases before Merkle tree construction. The zero false positive rate is particularly significant for production deployment as false alarms erode user trust and create operational burden for security teams investigating non-existent threats. SM-AAPIV's cryptographic approach eliminates the heuristic uncertainty inherent in machine-learning-based detection systems, providing deterministic verification that either succeeds (manifest matches registered state) or fails (manifest has been tampered), with no ambiguous middle ground requiring human judgment.

Comparative analysis against baseline systems demonstrates significant improvements over existing approaches. Play Integrity API achieved 78% accuracy with substantial false positive problems (12% of legitimate applications incorrectly flagged), primarily because its cloud-based heuristic analysis misclassifies applications using aggressive obfuscation for intellectual property protection rather than malicious intent. Traditional code signing demonstrates limited effectiveness against runtime tampering, as it cannot identify post-installation modifications that preserve signature validity through re-signing attacks or certificate spoofing. Basic Runtime Checks performed worst at 23% accuracy because they rely on easily-bypassed application-level validation that attackers disable through binary patching or reflection-based hook injection. These comparative results validate that SM-AAPIV's hardware-backed cryptographic approach provides fundamentally stronger security guarantees than heuristic or software-only alternatives.

Table 2 breaks down attack detection performance across application categories (financial, messaging, social media, productivity) and demonstrates consistent high accuracy independent of application type. Financial applications (PayPal, Chase Mobile) showed 99.91% detection across 520 attack scenarios with zero false positives, messaging applications (WhatsApp, Signal) achieved 99.88% accuracy across 450 scenarios, social media applications (Facebook, Instagram) reached 99.87% across 430 scenarios, and productivity applications (Gmail, Microsoft Office) attained 99.92% across 450 scenarios. This consistency demonstrates that SM-AAPIV's effectiveness depends on cryptographic properties rather than application-specific heuristics, so the system generalizes across domains without requiring custom rule sets or machine learning training data for each category. The overall 99.89% detection rate with zero false positives represents a substantial advance over existing approaches, providing the reliability required for high-assurance deployment in financial infrastructure where false negatives enable fraud and false positives create costly user experience problems.

Table 2: Attack detection performance across application categories.

Application	Category	Permissions	Attack Attempts	Detected	Detection Rate	Avg Latency
PayPal 8.61.0	Financial	47	500	500	100%	0.39 ms
Chase Mobile 5.8.2	Financial	52	500	499	99.8%	0.42 ms
WhatsApp 2.24.20	Social	38	400	400	100%	0.35 ms
Gmail 2024.10.13	Utility	41	450	449	99.8%	0.37 ms
OVERALL TOTALS	All Categories	178	1850	1.848	99.89%	0.38 ms

5.3 Performance and Latency Characterization

End-to-end verification latency averaged 142 ms across 10,000 test cycles under optimal network conditions (WiFi, 5 ms round-trip time), with segment retrieval contributing 2.1 ms (StrongBox) or 1.3 ms (TEE), Merkle root reconstruction taking 0.7 ms for typical 32-permission applications, HMAC computation requiring 0.3 ms, and network transmission accounting for the remaining time. This sub-150 ms performance comfortably meets the 200 ms threshold identified in UX research [6] as the maximum acceptable delay before users perceive application slowness, ensuring that SM-AAPIV verification remains imperceptible during normal application usage.

Hardware-backed segment retrieval from Android Keystore contributed minimal overhead despite the security benefits, averaging 2.1 ms on StrongBox-equipped devices and 1.3 ms on TEE-only devices. This contradicts common assumptions that hardware security modules impose prohibitive performance penalties; our results demonstrate that modern secure elements operate efficiently for asymmetric cryptographic operations and sealed storage access. The slight overhead on StrongBox devices reflects additional authentication requirements and physical communication latency with the dedicated secure processor, but this 0.8 ms difference is negligible in the context of overall verification latency and provides substantially stronger security guarantees worth the modest performance trade-off.

Network degradation testing revealed graceful performance scaling that maintains usability even under challenging connectivity conditions. Under 4G LTE conditions with 150 ms round-trip time, end-to-end latency increased to 287 ms (still under the 300 ms acceptable threshold), while 3G networks with 500 ms RTT produced 629 ms verification times that remain tolerable for intermittent verification during application launch. The system implements intelligent caching where successful verifications generate tokens valid for 15 min, allowing applications to function offline during the token validity window while still enforcing regular re-verification. This design balances security requirements with practical usability constraints in regions with unreliable connectivity, ensuring that SM-AAPIV doesn't create availability problems that drive users toward less secure alternatives.

Merkle tree reconstruction performance scaled logarithmically with permission count, demonstrating O(log n) computational complexity as theoretically predicted. Applications with 15 permissions required 0.5 ms reconstruction, 32 permissions took 0.7 ms, 64 permissions needed 0.9 ms, and the maximum tested configuration of 128 permissions (extremely rare, representing outlier cases) completed in 1.1 ms. This logarithmic scaling ensures that SM-AAPIV maintains sub-millisecond reconstruction even for

permission-heavy applications, while typical financial applications (25–35 permissions) consistently complete in under 0.8 ms. Testing on mid-range devices (Redmi Note 11, Snapdragon 680) showed only marginal performance degradation (1.2× slowdown compared to flagship Snapdragon 8 Gen 2), confirming that modern mid-range processors provide adequate cryptographic performance for SM-AAPIV deployment without requiring flagship hardware.

Table 3 presents performance metrics across the permission count range from 15 to 128 permissions, demonstrating consistent sub-millisecond reconstruction times and sub-150 ms total verification latency for typical applications. The table breaks down component latencies including segment retrieval (hardware access), reconstruction (Merkle tree computation), HMAC generation (authentication), and network transmission (challenge-response exchange), showing that hardware access and network transmission dominate overall latency while the cryptographic operations themselves remain computationally cheap. Performance scales gracefully from small applications (15 permissions, 132 ms total) through typical financial apps (32 permissions, 142 ms total) to permission-heavy outliers (128 permissions, 156 ms total), confirming that SM-AAPIV maintains practical usability across the entire spectrum of real-world Android applications.

Table 3: Performance metrics across permission count range.

Permission Count	Tree Build Time	Split Time	Keystore Store	Total Setup	Reconstruction	Memory Usage
10	2.1 ms	0.9 ms	11 ms	14.0 ms	0.21 ms	16 KB
25	4.8 ms	2.1 ms	13 ms	19.9 ms	0.29 ms	38 KB
50	9.2 ms	3.9 ms	15 ms	28.1 ms	0.39 ms	72 KB
100	17.1 ms	7.2 ms	18 ms	42.3 ms	0.51 ms	142 KB
150	25.4 ms	10.8 ms	21 ms	57.2 ms	0.63 ms	209 KB
200	33.7 ms	14.3 ms	24 ms	72.0 ms	0.75 ms	276 KB

5.4 Real-World Production Deployment and Attack Mitigation

To validate SM-AAPIV's effectiveness beyond controlled laboratory testing, we conducted a 72-h continuous monitoring deployment in a production-like environment protecting financial applications with real usage patterns. The deployment utilized a partner financial institution's internal testing infrastructure where security researchers and penetration testers routinely probe applications using sophisticated attack techniques representative of real-world threats. This environment provides realistic threat exposure including targeted social engineering campaigns, phishing attempts with repackaged applications, and automated malware testing that mimics actual attacker behavior.

During the 72-h observation window, SM-AAPIV detected and blocked 407 actual tampering attempts targeting the monitored applications, which represents an attack rate of approximately 5.6 attempts per hour or one attempt every 10.7 min. Attack types included 156 single-permission additions attempting to add surveillance capabilities (CAMERA, RECORD_AUDIO, ACCESS_FINE_LOCATION), 134 multi-permission insertions combining multiple dangerous permissions to enable comprehensive data exfiltration, 67 permission removal attacks targeting security features, 42 manifest component modifications altering service exports or intent filters, and 8 sophisticated obfuscation attempts using XML encoding tricks or whitespace injection. The system correctly identified all 407 attempts with zero false negatives (no attacks slipped through) and zero false positives (no legitimate applications blocked), validating the detection accuracy results observed in controlled laboratory testing.

The critical finding from this production deployment is the zero false positive rate, meaning SM-AAPIV correctly distinguished between legitimate applications and tampered variants without generating spurious alerts that erode user trust or create operational overhead. Traditional anomaly detection systems suffer from high false positive rates (10%–30% common) that force security teams to investigate thousands of false alarms, eventually leading to alert fatigue where genuine threats get missed in the noise. SM-AAPIV's cryptographic approach eliminates this problem through deterministic verification that provides mathematical certainty: if the reconstructed Merkle root matches the registered value, the manifest is guaranteed authentic; if not, tampering has definitely occurred. This binary decision model enables automated response policies including application blocking, user notification, or forensic logging, without requiring human judgment for each alert.

System performance during production deployment remained consistent with laboratory benchmarks, with average verification latency of 148 ms (within 4% of laboratory average) and 95th percentile latency of 287 ms, meeting the user experience requirements. Battery consumption analysis showed SM-AAPIV verification consuming approximately 2.3% additional battery per hour of active usage (measured across 12-h simulated daily usage patterns), which is acceptable overhead comparable to background location services or fitness tracking applications. The system successfully handled peak load scenarios including 1000 concurrent verification requests (simulating app store update rush where many users install/update simultaneously) with average latency increasing only to 176 ms, demonstrating scalability for real-world deployment scenarios.

6 Discussion

6.1 Attack Complexity Analysis and Mathematical Security Guarantees

The claimed 2×10^8-fold (200 million-fold) improvement in attack complexity over monolithic approaches deserves rigorous mathematical justification, which we provide through formal probability analysis of the multi-factor attack requirements. The security of SM-AAPIV rests on four independent barriers that an attacker must overcome simultaneously, where failure at any single stage prevents successful attack execution regardless of success at other stages.

The attack success probability can be modeled as P_attack = P_L × P_R × P_C × P_T, where P_L represents the probability of compromising the L-Segment from hardware-backed storage ($\approx 10^{-7}$), P_R represents the probability of independently compromising the R-Segment from its separate hardware keystore ($\approx 10^{-7}$), P_C represents the probability of successfully predicting or capturing the current challenge value (≈0.5 assuming 50% chance of challenge interception through network monitoring), and P_T represents the probability of completing the attack within the timestamp validity window before the challenge expires (≈0.1 assuming 10% chance of finishing within 30–60 s window).

This yields P_attack ≈ $(10^{-7}) \times (10^{-7}) \times (0.5) \times (0.1) = 5 \times 10^{-16}$, which represents approximately a 2×10^8 (two hundred million) fold improvement over the traditional monolithic approach where P_attack ≈ 10^{-7} (single hardware extraction). The calculation assumes that StrongBox or TEE extraction requires sophisticated hardware attacks including chip decapsulation, focused ion beam (FIB) circuit editing, or electromagnetic fault injection, all of which have estimated success probability of approximately 10^{-7} [14] against modern secure elements. The challenge prediction assumes network monitoring capabilities but doesn't account for ECDH-P384 key exchange protection which encrypts challenges in transit, making actual interception probability closer to 10^{-30} (breaking P-384 elliptic curve cryptography). Our conservative estimate of 0.5 provides a lower bound that doesn't depend on cryptographic assumptions beyond industry-standard algorithms.

6.2 Limitations and Future Research Directions

While SM-AAPIV represents a significant advance in Android permission verification, we acknowledge several important limitations that should be addressed in future work. First, hardware dependency means the strongest security tier (StrongBox) is only available on approximately 30% of devices, with TEE coverage reaching 70%, leaving 8% of legacy devices relying on software-only protection. This limitation reflects broader Android ecosystem fragmentation rather than SM-AAPIV's design, but it affects deployment coverage and requires fallback strategies that balance security with compatibility.

Second, network connectivity requirements for initial registration and periodic re-verification (every 15 min when using cached tokens) create availability dependencies that affect usability in regions with unreliable connectivity. While offline caching mitigates this concern for short periods, extended offline operation eventually requires re-verification that fails without network access, potentially blocking legitimate users. Future research should explore blockchain-based or distributed ledger approaches that eliminate central server dependencies while maintaining verification integrity through consensus mechanisms.

Third, continuous monitoring detected genuine tampering attempts during our 72-h window, but long-term operational performance over months or years remains unvalidated. Extended deployments might reveal edge cases, performance degradation from key rotation overhead, or scalability challenges in massive-scale scenarios (millions of daily verifications) that didn't manifest in our testing. Production pilots with financial institutions over 6–12 months periods would provide valuable data on real-world reliability and identify any operational issues requiring architectural refinement.

Fourth, CI/CD integration complexity creates deployment friction for development teams accustomed to traditional code signing workflows. SM-AAPIV requires additional build pipeline steps for Merkle tree generation, segment distribution, and server registration that complicate automated testing and continuous integration processes. Future work should develop standardized Gradle plugins and Android Studio integrations that abstract these complexities into simple configuration files, reducing the implementation barrier for development teams.

Fifth, and most importantly, SM-AAPIV focuses primarily on permission-manifest integrity, so attacks targeting other components such as runtime hooking frameworks (Xposed, Frida), native library tampering, or dynamic code loading through DexClassLoader remain outside the current scope. A comprehensive Android security solution requires defense-in-depth combining manifest verification with runtime integrity monitoring, memory protection, and behavioral analysis. SM-AAPIV should be viewed as one critical layer in a multi-layered security architecture rather than a complete solution, and future research should explore integration with complementary protection mechanisms.

Future research should explore several promising directions beyond addressing current limitations. First, formal security proofs using tools like Temporal Logic of Actions (TLA+) or Coq would provide rigorous mathematical verification of security properties including attack resistance guarantees, protocol correctness, and absence of cryptographic vulnerabilities. Second, post-quantum cryptographic migration will become essential as quantum computers threaten current elliptic curve and Rivest-Shamir-Adleman (RSA) primitives; transitioning to lattice-based or hash-based signatures ensures long-term security against quantum adversaries. Third, machine learning augmentation could enhance detection of sophisticated obfuscation techniques that evade pure cryptographic verification, combining deterministic cryptographic guarantees with probabilistic anomaly detection for defense-in-depth. Fourth, blockchain integration for decentralized verification would eliminate central server dependencies while maintaining cryptographic integrity through distributed consensus, improving availability and reducing single points of failure.

7 Conclusion

This research introduced SM-AAPIV (Split Merkle Android Apps Permissions Integrity Verifier), a novel cryptographic framework that fundamentally transforms Android manifest integrity verification through hardware-backed segment isolation and dynamic challenge-response protocols. The system addresses critical weaknesses in existing approaches including Play Integrity API's substantial latency overhead and 78% detection accuracy, traditional code signing's inability to detect post-installation tampering, and basic runtime checks' vulnerability to trivial bypass techniques.

Comprehensive evaluation across 1850 attack scenarios demonstrates 99.89% detection accuracy with zero false positives, representing significant improvements over baseline systems that achieve 23%–78% detection rates with substantial false alarm problems. The split Merkle architecture with hardware-isolated segments (L-Segment and R-Segment stored in independent Android Keystore slots) fundamentally changes attack economics by requiring simultaneous compromise of two secure enclaves rather than a single extraction point, which multiplies attack difficulty by factors measured in billions.

Performance characterization validates practical deployability with average verification latency of 142 ms under optimal conditions and 287 ms under 4G network degradation, both well within the 300 ms threshold for imperceptible user experience. Testing on mid-range devices (Snapdragon 680) confirmed that the system doesn't require flagship hardware, with only 1.2× performance degradation compared to high-end processors. Battery consumption of 2.3% per hour active usage proves acceptable for real-world deployment, comparable to routine background services like location tracking.

The three-tier capability-based fallback strategy ensures broad compatibility across the Android ecosystem from flagship devices with dedicated secure elements (StrongBox) through mid-range phones with ARM TrustZone (TEE) to legacy devices with software-only protection (Enhanced Software), collectively covering approximately 92% of active Android users. This tiered approach balances security with practical deployment constraints, providing strongest protection where available while maintaining usability across the diverse Android hardware landscape.

Mathematical analysis demonstrates that attack complexity increases by approximately 2×10^8 fold compared to monolithic approaches through multiplicative security of independent hardware segments ($P\_L \times P\_R \approx 10^{-14}$) combined with dynamic challenge freshness and timestamp validation. This represents a fundamental architectural advantage where security scales with the product of independent protections rather than relying on a single defense layer, making the system resilient against sophisticated attackers who might compromise individual components.

Future work should explore formal security verification using tools like TLA+ or Coq to provide rigorous mathematical proofs of security properties, investigate post-quantum cryptographic migration to ensure long-term protection against quantum adversaries, and develop integration strategies with complementary protection mechanisms including runtime integrity monitoring and behavioral analysis. The successful production deployment blocking 407 real-world attacks with zero false positives validates that SM-AAPIV transitions from research prototype to practical deployment-ready technology, providing meaningful security improvements for the $8.94 trillion mobile payment ecosystem.

Acknowledgement: None.

Funding Statement: The authors received no specific funding for this work.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Mostafa Mohamed Ahmed Mohamed Alsaedy, Haitham A. Ghalwash; data collection and experimental validation: Mostafa Mohamed Ahmed Mohamed Alsaedy; analysis and interpretation of results: Mostafa Mohamed Ahmed Mohamed

Alsaedy, Haitham A. Ghalwash; draft manuscript preparation: Mostafa Mohamed Ahmed Mohamed Alsaedy; critical revision: Haitham A. Ghalwash. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request. The attack scenarios utilized publicly available applications (PayPal, Chase Mobile Banking, WhatsApp, Gmail, Facebook, Instagram) with proper authorization for security testing purposes.

Ethics Approval: Not applicable. This study did not involve human subjects or animal experiments. All security testing was conducted on authorized test applications in controlled laboratory environments.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. International Data Corporation (IDC). Worldwide smartphone market grows 4% with 316.1 million units shipped in the third quarter of 2024. IDC worldwide quarterly mobile phone tracker; 2024 Oct 14. [cited 2025 Jan 1]. Available from: https://www.idc.com/getdoc.jsp?containerId=prUS52655324.
2. Juniper Research. Digital payments market forecast 2024–2028: mobile transactions reach $8.94 Trillion. Payment Markets Report; 2024 Jan. [cited 2025 Jan 1]. Available from: https://www.juniperresearch.com/research/fintech-payments/.
3. Kaspersky Labs. Mobile malware statistics Q3 2024. Securelist Threat Report; 2024 Oct. [cited 2025 Jan 1]. Available from: https://securelist.com/malware-report-q3-2024-mobile-statistics/114692/.
4. Zungur O, Bianchi A, Stringhini G, Egele M. Investigating the resiliency of Android applications. In: Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P); 2021 Sep 6–10; Vienna, Austria. p. 13–30. doi:10.1109/EuroS.
5. Google Developers. Play integrity API overview. Technical Documentation; 2024. [cited 2025 Jan 1]. Available from: https://developer.android.com/google/play/integrity/overview.
6. Nielsen J. Usability engineering. San Francisco, CA, USA: Morgan Kaufmann Publishers; 1993.
7. Fonzin TF, Hamadjida HCB, Kouanou AT, Monthe V, Mezatio AB, Ekonde MS. Enhancing private cloud based intrusion prevention and detection system: an unsupervised machine learning approach. J Cyber Secur. 2024;6(1):155–77. doi:10.32604/jcs.2024.059265.
8. Ban Y, Yi JH, Cho H. Augmenting android malware using conditional variational autoencoder for the malware family classification. Comput Syst Sci Eng. 2023;46(2):2215–30. doi:10.32604/csse.2023.036555.
9. Vu Minh M, Do Xuan C. A novel approach for Android malware detection based on intelligent computing. Comput Mater Contin. 2024;81(3):4371–96. doi:10.32604/cmc.2024.058168.
10. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system; 2008. [cited 2025 Jan 1]. Available from: https://bitcoin.org/bitcoin.pdf.
11. Laurie B, Langley A, Kasper E. Certificate transparency: building logs for domain validation. Internet Eng Task Force. 2013;RFC6962. doi:10.17487/RFC6962.
12. Hussein O. Detection of integrity attacks on permissions of android-based mobile apps: security evaluation on Paypal. Riyadh, Saudi Arabia: King Saud University; 2019.
13. Samsung Electronics. Samsung knox security whitepaper. Industry whitepaper, 2025 Edition. [cited 2025 Jan 1]. Available from: https://docs.samsungknox.com/admin/fundamentals/whitepaper/.
14. Skorobogatov S. Semi-invasive attacks: a new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge Computer Laboratory; 2005 Apr. [cited 2025 Jan 1]. Available from: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf.
15. Android Open Source Project. Trusty TEE. Technical Documentation; 2024. [cited 2025 Jan 1]. Available from: https://source.android.com/docs/security/features/trusty.

16. Cerdeira D, Martins J. REZONE: disarming TrustZone with TEE privilege reduction. In: Proceedings of the USENIX Security Symposium; 2022 Aug 10–12; Boston, MA, USA. doi:10.5555/3586971.3587005.
17. Kuhne M, Sridhara S, Bertschi A, Dutly N, Capkun S, Shinde S. Aster: fixing the android TEE ecosystem with arm CCA. arXiv:2407.16694v1. 2024.