THE ZERO SECRETS ARCHITECTURE

A MODERN FOUNDATION FOR MOBILE APP & API SECURITY IN THE AGE OF AGENTIC AI

approov

Contents

Executive Summary 3

1. The Threat Landscape Has Changed 4

1.1 APIs Are Now the Primary Attack Surface 4
1.2 Agentic AI is a New Class of Adversary 4
1.3 The Defender's Old Toolbox Is Empty 4

2. Why Zero Secrets is Now Urgent 6

2.1 What Counts as a Secret 6
2.2 The Three Extraction Vectors 7
2.3 The Backend Consequences 7

3. The Four Pillars of a Zero Secrets Architecture 8

3.1 Dynamic Certificate Pinning 8
3.2 Cross-Platform Mobile App Attestation 9
3.3 Runtime Secrets Protection 10
3.4 Short-Lived JWTs for API Access 11

4. Defeating Replay Attacks: Token Binding and Message Signing 13

4.1 Token Binding 13
4.2 Message Signing 13
4.3 Starving the Agent 14

5. Adoption Roadmap 15

Phase 1: Inventory and Baseline 15
Phase 2: Establish Attestation 15
Phase 3: Replace Static Pinning and Long-Lived Tokens 15
Phase 4: Remove Secrets 16
Phase 5: Protect the Token and the Request 16
Phase 6: Operate, Measure, Iterate 16

6. The CISO Checklist 17

7. Conclusion 19

About Approov 19

The Zero Secrets Architecture | A Modern Foundation for Mobile App and API Security in the Age of Agentic AI

Prepared for Chief Information Security Officers and Mobile Security Architects

Published by Approov Limited | May 2026

Executive Summary

Mobile applications and the APIs they consume have become the operational backbone of modern digital business. They are also the most exposed and least defended layer of the enterprise. The mobile binary runs on hardware the security team does not own, in environments the security team cannot inspect, and increasingly, it is being targeted not by human attackers with manual tooling but by autonomous AI agents that can reverse-engineer, probe, and abuse APIs at machine speed.

For more than a decade, the dominant approach to mobile security has rested on a single, flawed assumption: that secrets embedded in the application binary can be hidden well enough to deter attackers. That assumption has always been weak. In an environment where AI assistants can de-obfuscate code in seconds, where hooking frameworks can extract credentials from process memory at runtime, and where behavioral signals can be convincingly mimicked by AI-driven traffic generators, that assumption has now collapsed entirely.

This paper argues for a fundamental shift in mobile and API security strategy: the adoption of a Zero Secrets Architecture. The premise is straightforward. If a secret is not present in the application binary, on disk, or persistently in memory, it cannot be extracted from the application binary, from disk, or from memory. Trust shifts from the artifact to the environment, from the credential to the verified runtime, and from static authentication to dynamically delivered, short-lived authorization.

What This Paper Covers

- Why traditional defenses—code obfuscation, embedded secret hiding, certificate pinning baked into the binary, and behavioral analytics—no longer provide meaningful protection against agentic AI threats.
- How the Zero Secrets model removes attackable material from the client, replacing static credentials with verified, just-in-time authorization tied to a specific, attested app instance.
- The four pillars of a modern mobile and API defense: dynamic certificate pinning, cross-platform mobile app attestation, runtime secrets protection, and short-lived JWTs for API access.
- Advanced techniques—Token Binding and Message Signing—that are now necessary, not optional, to defeat replay attacks executed by automated agents.
- A practical adoption roadmap, integration patterns, and a CISO-level checklist for evaluating the maturity of an organization's mobile and API security posture.

The Bottom Line for the CISO

An AI agent that cannot find a key, cannot replay a token, and cannot impersonate an app instance has nothing to attack. The Zero Secrets Architecture is the most direct way to deny agentic AI the raw material it needs to abuse APIs at scale. It is also a forcing function for hygiene that organizations should have adopted years ago.

©2026 Approov Limited | All rights reserved | Version 1.0

1. The Threat Landscape Has Changed

Three trends have collided over the past two years to render the dominant mobile security playbook obsolete. They are not separate problems. Together they define a new operating environment that defenders must plan for, not an emerging concern that can be deferred.

1.1 APIs Are Now the Primary Attack Surface

Mobile applications are, in practical terms, thin clients. The valuable data, the regulated PII, the payment flows, the account management functions, and the business logic that produces revenue all live behind APIs. The mobile app is simply the most distributed, least defensible client the enterprise exposes to those APIs.

This concentration of value behind APIs has not gone unnoticed. Industry data through the first half of 2026 shows that organizations are reporting API growth of 51 to 100 percent year over year, and a majority of security leaders report that legacy controls—web application firewalls, network-layer rate limiting, signature-based bot detection—are no longer effective against the threats they now face. The mobile channel feeds these APIs, and the mobile channel is where the trust boundary is weakest.

1.2 Agentic AI is a New Class of Adversary

The arrival of agentic AI—autonomous systems that plan, use tools, maintain memory across sessions, and execute multi-step workflows without human oversight—represents a categorical change in the threat model. Earlier generations of automated attacks were essentially scripts: brittle, predictable, and detectable through behavioral heuristics. Agentic systems are none of those things.

An agent given the goal of abusing a mobile API can decompile a published binary, identify embedded keys and endpoints, generate the necessary client code, defeat simple anti-tampering checks, and begin issuing requests at scale—all without a human in the loop and at a cost approaching zero. The same agent can iterate on its approach when blocked, A/B test evasions, and coordinate across thousands of synthetic identities. The attack model that defenders must now plan for assumes a tireless adversary with the analytical capability of a senior reverse engineer and the operational capacity of a cloud workload.

Why Agentic AI Changes the Calculus

- Reverse engineering that previously took a skilled attacker days now takes an agent minutes.
- Behavioral patterns that previously distinguished bots from humans can now be synthesized convincingly—mouse movements, typing cadence, navigation paths, dwell times.
- Attacks scale horizontally across cheap compute using bot farms and virtualizations. The cost of running ten thousand simultaneous synthetic users is no longer a meaningful constraint.
- Agents can coordinate. A compromised credential extracted from one app instance can be replayed across an entire orchestrated fleet within seconds.

1.3 The Defender's Old Toolbox Is Empty

Three controls in particular, long considered staples of mobile security programs, no longer carry their weight against this combined pressure of API exposure and agentic capability.

©2026 Approov Limited | All rights reserved | Version 1.0

Code Obfuscation

Obfuscation—renaming symbols, control-flow flattening, string encryption, and similar transformations—was always a tactic to block low sophistication attacks rather than a holistic defense. It increased the time required for a human reverse engineer to understand a binary; it never prevented understanding. With AI-assisted decompilation and de-obfuscation now widely available, that time-cost benefit has effectively disappeared. An agent can pass an obfuscated binary through a decompiler, ask a model to rename variables and reconstruct intent, and have a working analysis in the time it takes a human to brew a cup of coffee. Obfuscation is the security equivalent of erasing the street names on a map: it might frustrate a tourist, but the locals know exactly where to go.

Hiding Secrets in the Binary

Every mobile application contains secrets—API keys, third-party tokens, signing keys, configuration data, sensitive endpoint URLs. The historical response was to encrypt these strings, split them across multiple files, generate them at runtime from device-derived inputs, or store them in native code believed to be harder to inspect. None of these techniques survives contact with a determined attacker, and certainly not with a determined agent. Static analysis recovers obfuscated strings; dynamic analysis with hooking frameworks captures them at the moment of use; memory inspection extracts them from the heap. A secret that exists on the device is a secret that will be extracted from the device. The only durable defense is for the secret not to be there in the first place.

Behavioral Signals

For years, defenders relied on behavioral analytics—patterns of touch, scroll velocity, navigation sequences, sensor readings—to distinguish legitimate users from automated traffic. The premise was that human behavior is sufficiently complex and idiosyncratic that machines could not credibly reproduce it. Agentic AI has invalidated that premise. A modern agent can drive a real device, generate plausible sensor traces, mimic interaction patterns derived from human telemetry, and pace its actions to look entirely human. Behavioral signals retain some narrow utility for detecting unsophisticated bots, but they are no longer a reliable boundary for high-value APIs. Treating them as one creates a false sense of safety while the real attack passes through.

The Honest Assessment

Code obfuscation, embedded-secret hiding, and behavioral analytics are not worthless. They each impose some friction, and friction has value at the margin. They are, however, no longer load-bearing. A mobile and API security program that depends on them as primary controls is a program that has not yet adjusted to the actual threat environment.

©2026 Approov Limited | All rights reserved | Version 1.0

2. Why Zero Secrets is Now Urgent

The case for removing secrets from the mobile application is not new. Practitioners have understood the risk of embedded credentials for as long as mobile applications have existed. What is new is that the cost of leaving secrets in place has risen sharply while the cost of removing them has fallen, and the gap between the two has now made action urgent rather than aspirational.

2.1 What Counts as a Secret

Before discussing how to remove secrets, it helps to be specific about what is meant by the term. A secret is any data point intended to be known only to authorized parties whose disclosure would enable unauthorized access, impersonation, or abuse. In a typical mobile application this includes more than most teams initially realize.

Category	Examples	Impact if Disclosed
API Keys	First-party API keys; backend access tokens	Direct API impersonation; ability to issue authenticated requests at attacker scale
Third-Party Credentials	Keys for analytics, messaging, mapping, payment, observability and other integrated services	Abuse of paid services on the developer's account; data exfiltration through third parties; quota exhaustion
Infrastructure Access	Cloud service credentials; database connection strings; SDK licence keys	Direct access to backend infrastructure, often outside the API trust boundary
Sensitive URLs	Internal service endpoints; staging hosts; private resource locators	Discovery of attack surface that was assumed to be unpublished
Public Cryptographic Keys	Pinning material; signature verification keys	Replay attacks if rotated keys are unavailable; weakened transport trust
Private Cryptographic Keys	Signing keys; client certificates; identity material	Full identity impersonation; ability to forge requests indistinguishable from a legitimate client
Hashes and Salts	Stored together for client-side checks	Reversibility through known-plaintext or rainbow-table attacks once the salt is recovered
Configuration Data	Environment files; build manifests; feature flags exposing internals	Reconnaissance, environment fingerprinting, attack tailoring

A practical Zero Secrets program treats every entry in this list as an artifact that must not exist on the device in extractable form. Cryptographic public keys used purely for verification are a partial exception, but only when they

©2026 Approov Limited | All rights reserved | Version 1.0

can be rotated dynamically without releasing a new app version, since static pinned keys become liabilities at the moment they need to change.

2.2 The Three Extraction Vectors

Any secret on the device is exposed to three independent extraction techniques. Defending against one without defending against the others provides no meaningful protection.

Static Analysis and Reverse Engineering

The published binary is the most accessible artifact in the entire mobile ecosystem. Anyone can download it, decompile it with freely available tooling, and inspect its contents at leisure. Strings, embedded resources, native libraries, and the application logic itself are all visible. Obfuscation slows this analysis but does not prevent it, and AI-assisted decompilation has compressed the timeline from days to minutes.

Man-in-the-Middle Interception

Even when secrets are not embedded in the binary, they are typically transmitted to and from the backend over TLS. An attacker who can position a proxy between the app and the network—trivially achievable on a controlled device—can observe these flows unless the application validates the server certificate against a trusted reference. The historical solution, certificate pinning, has well-known operational problems that have led many teams to weaken or remove it. We address those problems and the modern solution in Section 3.

Dynamic Analysis at Runtime

Even a secret that is not in the binary, not transmitted in plaintext, and not stored at rest must, at some point, exist in process memory while it is being used. Hooking frameworks such as Frida, instrumented runtimes, and rooted or jailbroken devices give attackers unrestricted access to that memory. A secret loaded into a buffer, decrypted into a variable, or passed as an argument to a system call can be captured at the moment of use. The conclusion is unavoidable: secrets that need to exist on the device must exist there for the shortest possible time, in the smallest possible quantity, and only on devices that have been verified to be uncompromised.

2.3 The Backend Consequences

The Compounding Effect

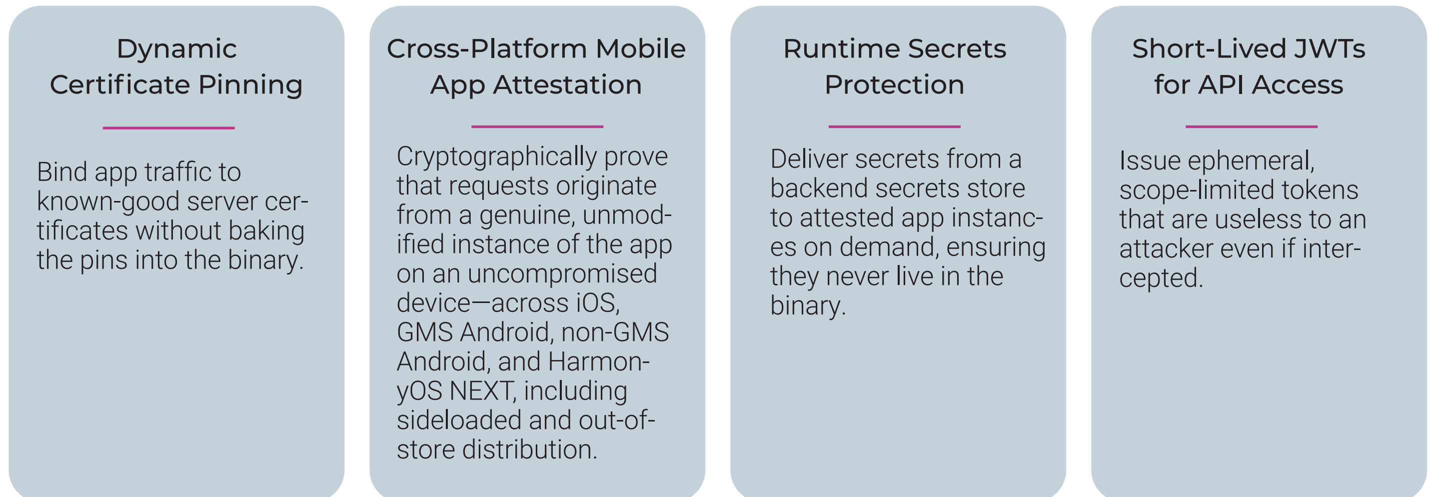
Each of the three extraction vectors is independently sufficient to compromise an embedded secret. An attacker only needs one to succeed. A defender must close all three. This asymmetry is precisely why the Zero Secrets premise—removing the secret from the device entirely—is the only durable answer.

A compromised mobile app does not stay a mobile problem. The credentials embedded in the app authenticate to backend services, and once those credentials are extracted, the attack moves to the backend at scale. The blast radius typically includes API abuse, in which the attacker issues authenticated requests at volumes the legitimate app could never produce; credential stuffing, in which extracted authentication material is replayed against other endpoints; large-scale data scraping, which converts public APIs into bulk extraction tools for the underlying data; denial of service or wallet draining, where authenticated abuse exhausts paid quotas, rate limits, or compute budgets; and reputational and regulatory exposure, since the breach is traced back to the published mobile artifact regardless of where the abuse ultimately occurred. The mobile binary, in this sense, is the soft underbelly of the entire API surface. Hardening the perimeter while leaving the binary exposed is a familiar mistake with an increasingly expensive bill.

©2026 Approov Limited | All rights reserved | Version 1.0

3. The Four Pillars of a Zero Secrets Architecture

A Zero Secrets Architecture rests on four technical capabilities that work together. None is sufficient on its own; each addresses a specific failure mode of the others, and together they form the minimum viable foundation for modern mobile and API defense.



3.1 Dynamic Certificate Pinning

Certificate pinning is the practice of validating that the server presenting a TLS certificate matches a specific, expected reference, rather than accepting any certificate signed by a trusted certificate authority. It exists because the public CA system, while convenient, allows any of dozens of authorities to issue a valid certificate for any domain. An attacker who can compel or compromise a single CA, or who has a corporate CA installed on a controlled device, can present a TLS certificate the operating system trusts. Without pinning, the app accepts the impostor and the channel is open for inspection.

Static pinning—hardcoding pin values into the application binary—has been the traditional implementation. It is also the reason many teams have weakened or removed pinning entirely. When a server certificate must be rotated for any reason—expiry, suspected compromise, infrastructure migration—the pinned values must be updated, and that update can only reach users through a new app release. Update propagation in mobile is slow and incomplete; some users never update. The result is a choice between rotating certificates and breaking the app for a meaningful slice of the user base, or not rotating and accepting the security debt. Many teams quietly chose the latter.

The Modern Approach

Dynamic certificate pinning solves this trade-off by separating the act of pinning from the act of releasing. The current set of valid pins is delivered to the application from a trusted control service at runtime, on a short cadence. When a server certificate is rotated, the new pin set propagates to the entire installed base within minutes, at the next attestation cycle. There is no app release required, no broken users, no reason to weaken the control.

This approach also enables behavior that static pinning cannot match. Pins can be rotated proactively on a schedule, narrowing the window of any individual pin's exposure. Compromise of a server certificate, once detected, can be contained almost immediately. Different environments—production, staging, and partner integrations—can carry different pin sets without code branches.

©2026 Approov Limited | All rights reserved | Version 1.0

CISO Perspective

A program that does pinning statically is, in most cases, doing pinning poorly. The operational pain has typically pushed teams toward weak fallbacks, exception lists, or outright disablement. Dynamic pinning is the only form of pinning that is both effective in production and operationally sustainable across the lifecycle of a real application portfolio.

3.2 Cross-Platform Mobile App Attestation

Attestation answers a simple question with significant consequences: is the request actually coming from an unmodified instance of our published app, running on a device that has not been compromised, in an environment that has not been instrumented for analysis? Without an answer to this question, every other control rests on an assumption the defender cannot verify.

Mobile app attestation is the cryptographic proof that the request originated from a trusted client. It must verify not only that the calling code is the published binary—a signature check, in essence—but also that the runtime environment is intact: that the device is not rooted or jailbroken in a way that would allow tampering, that the app is not running under a debugger or hooking framework, that the binary has not been modified since signing, and that the request is not being replayed from an emulator, a repackaged clone, or a synthetic environment created specifically to abuse the API.

Why Cross-Platform Matters Globally

Both Apple and Google provide platform-native attestation services—App Attest and Play Integrity, respectively. These are useful primitives but not complete solutions, and they have a more significant blind spot than is widely acknowledged: they cover only a fraction of the global mobile install base.

A serious mobile portfolio in 2026 spans considerably more than iOS and Google-services Android. Three additional ecosystems matter, and any of them can represent the majority of users for a given application depending on where its customers live.

HarmonyOS NEXT

Huawei's HarmonyOS has surpassed iOS in mainland China, with roughly 18 percent share in the first half of 2026, and is now the third-largest mobile operating system globally. HarmonyOS NEXT (the "pure-blood", fully self-developed lineage that began shipping at scale through 2025 and 2026) departs from Android entirely: it has its own microkernel, its own development stack (ArkTS and DevEco Studio), its own application package format, and its own distribution channel (Huawei AppGallery). For applications targeting China—and increasingly the markets Huawei is expanding into across Southeast Asia, the Middle East, and parts of Africa—HarmonyOS NEXT is not an edge case. It is a primary platform whose user base already exceeds iOS in its home market. Play Integrity does not run on it. App Attest does not run on it. A mobile and API security strategy that has nothing to say about HarmonyOS NEXT has nothing to say about an entire category of users.

Non-GMS Android

A substantial share of Android devices ship without Google Mobile Services. This is the default configuration in mainland China, where Google services are blocked entirely; it is common in India, where price-sensitive devices increasingly use AOSP-based alternatives; it is common across Latin America, the Middle East, and Africa for the same economic reasons; and it is universal on Huawei devices outside China since 2019. These devices use alternative app stores—Huawei AppGallery, Xiaomi GetApps, Samsung Galaxy Store, Amazon Appstore, APKPure, Aptoide, and a long tail of regional marketplaces—and they cannot call Play Integrity at all, because the underlying Google Play Services framework simply is not present on the device.

For a global application, the practical consequence is that platform-native Android attestation provides no signal

©2026 Approov Limited | All rights reserved | Version 1.0

whatsoever for the meaningful share of legitimate users who happen to be on a non-GMS device. Treating those users as untrusted by default is not an option; the result would be a broken application across major markets. Treating them as trusted by default is also not an option; the result is an unguarded channel that an adversary will quickly find and exploit.

Sideloaded and Out-of-Store Distribution

Outside North America and Western Europe, sideloading is a normal user behavior, not an exception. Users obtain APKs directly from publisher websites, from third-party stores, from social-media links, and from local app marketplaces. Enterprise distribution—internal apps deployed through MDM, partner apps shared with field staff, white-label builds for downstream brands—compounds this further. Platform-native attestation services were designed primarily for store-distributed apps and provide weaker, partial, or nonexistent guarantees for binaries delivered through these other channels.

A unified, cross-platform attestation layer is the only practical answer to this fragmentation. It applies the same trust decisions across iOS, GMS Android, non-GMS Android, and HarmonyOS NEXT; it produces the same proof format to backend services regardless of where the request originated; and it works across the cross-platform frameworks (Flutter, React Native, .NET MAUI, Capacitor, Unity) that engineering teams use to ship to multiple ecosystems from a single codebase. Crucially, it operates independently of platform-native attestation services, so an app distributed outside a public store, on a non-GMS Android variant, or on HarmonyOS NEXT receives the same protection as one distributed through the App Store or Play Store.

The Global Coverage Gap

Roughly one in five mobile users worldwide is on a device where neither App Attest nor Play Integrity provides usable trust signals. In China, that share is closer to three in five. A security architecture that quietly excludes those users—either by failing to defend their requests or by blocking them from the application entirely—is not a global architecture. It is a North-American-and-Europe architecture with a coverage problem the threat actors will reach long before the security team does.

What Attestation Enables

- Refusal of API requests that do not carry a valid attestation, eliminating impersonation by reverse-engineered or repackaged clients.
- Detection of compromised runtime environments, including rooted devices, hooking frameworks, instrumentation, and debugger attachment, before sensitive operations occur.
- Differentiated trust decisions: full access for attested instances, degraded or refused access for unattested or partially attested ones.
- A foundation for the other three pillars, each of which depends on knowing that the entity making the request is, in fact, the genuine app.

3.3 Runtime Secrets Protection

Once attestation is in place, the question of how to handle secrets becomes tractable. The operating principle is simple: a secret is delivered from a controlled backend to a verified app instance only at the moment it is needed, used, and discarded. It is not stored in the binary. It is not written to disk. It is not retained in memory beyond its narrow window of use. It is, in effect, ambient rather than embedded.

©2026 Approov Limited | All rights reserved | Version 1.0

Two Patterns, Both Zero Secrets

In practice there are two architectural patterns that achieve this outcome. They differ in where the secret is finally used, but both ensure the secret is never present in the application binary, and both depend on attestation to gate access.

Pattern A: The Proxy Pattern

In the proxy pattern, the mobile application never holds the third-party secret at all. Instead, the app authenticates to a controlled backend proxy using its attestation token, and the proxy uses its own securely stored secret to invoke the third-party service on the app's behalf. The response is returned to the app over the authenticated channel. The third-party credential remains exclusively on the server, where it can be rotated, monitored, and protected with conventional infrastructure controls. This pattern is the strongest of the two and should be preferred whenever the architecture allows it. Note that this pattern requires strong attestation to be secure, including advanced steps to protect the token such as Message Signing.

Pattern B: Runtime Secrets Delivery

Some integrations cannot be proxied—for example, when a third-party SDK must be invoked directly from the device, or when the third-party service does not support delegated authentication. For these cases, the secret is held in a backend secrets store and delivered to the verified app instance just in time, encrypted in transit, and used immediately. Delivery is gated by attestation: only an instance whose runtime is verified intact will receive the secret, and the delivery itself is a discrete, auditable event rather than a static configuration. The secret is not persisted on the device, not written to local storage, and the application's handling of it is bound to the smallest possible scope.

Aspect	Proxy Pattern	Runtime Secrets
Where the secret lives	Backend only	Backend at rest; in-memory on device only at moment of use
Where the secret is used	Backend (calls third party on app's behalf)	Mobile app (direct call to third party)
Best fit	First-party APIs and proxiable third parties	Direct-to-SDK integrations that cannot be proxied
Strength	Strongest—secret never touches the device	Strong when delivery is gated and lifetime is minimal
Operational considerations	Adds a backend hop; requires proxy infrastructure	Requires careful in-app handling; benefits from short rotation

3.4 Short-Lived JWTs for API Access

The fourth pillar addresses the moment of API access itself. Even with secrets removed from the binary and attestation in place, the API still needs to authorize the specific request in front of it. Long-lived bearer tokens—the historical default—are unsuited to this role. A token valid for hours or days is a token an attacker has hours or days to capture and replay; once compromised, it functions as a master key for the duration of its lifetime.

Short-lived JSON Web Tokens, issued on demand and bound to a specific app instance, are the modern replace-

©2026 Approov Limited | All rights reserved | Version 1.0

ment. They carry a few minutes of validity, encode narrow scopes, and are rotated continuously. Even if intercepted, the attacker has only a small window in which the token has any value, and that window closes automatically without operator intervention. Combined with attestation, the issuance of these tokens itself becomes a trust decision: only verified app instances receive them, and the token's claims encode the trust state at the moment of issuance.

Why This Matters Against Agentic Attackers

Agentic systems excel at finding and replaying captured credentials. A long-lived token leaked from any source—a debugger session on a controlled device, a misconfigured logging pipeline, a third-party SDK with telemetry the developer did not vet—becomes a durable asset that an agent can use, share, and combine with other captured material indefinitely. A short-lived token denies the agent that durability. Each request requires a fresh issuance, each issuance requires a fresh attestation, and the attestation requires the agent to continuously prove something it cannot prove: that it is a genuine app on a clean device.

The Combined Effect

Dynamic pinning ensures the channel is to the right server. Attestation ensures the request is from the right client. Runtime secrets ensure the right credentials are used without ever being embedded. Short-lived JWTs ensure that authorization is fresh, scoped, and useless if intercepted. Each pillar removes one of the assumptions an attacker would otherwise be able to exploit. Together they leave very little for an agent to work with.

©2026 Approov Limited | All rights reserved | Version 1.0

4. Defeating Replay Attacks: Token Binding and Message Signing

Even with the four pillars in place, a sophisticated agentic adversary will probe for the seams. The seam most worth examining is the period during which a legitimately issued token is in flight or in use. If an attacker can capture a valid token within its short lifetime, can the attacker then replay it from a different context to abuse the API? Without additional controls, the answer is often yes. With token binding and message signing, the answer becomes no.

4.1 Token Binding

Token Binding associates an issued token with the specific details known to the API service with the runtime context for which it was issued, such as a session key. The token is not a bearer credential that anyone in possession can use; it is usable only when presented to validate that the token was generated by the same context used to generate the hash. The proof is derived or assembled from data known to the server (such as a session key) and transferred via a validly attested API call or calls. It is added to a claim in the JWT Token. The API service can verify the claim during each API request .

The practical effect is that a token captured by an attacker—whether through a misconfigured proxy, a logging leak, or a hostile observer of the network—is unusable outside the binding context in which it was issued. The API service will detect that the current session key does not match the hashed key added to the token. Keys can also be assembled from multiple pieces of information, such as data from two different API endpoints, or a calculated seed plus a key. The attacker would need to have access to the key data and any operation done on the data before the hash is generated by obtaining it from its API source and recreating it. A valid attestation will show if the data was not assembled and hashed correctly.

4.2 Message Signing

Message Signing cryptographically proves the origin of the “message” - the API request - by signing it with a private key generated upon the first run of the app and stored securely on the device. Each API call includes a signature header, as well as the Approov token, which contains the installations’ public key. The server validates the signature before processing the request. A signed request that is captured and replayed against the API will fail validation, because the timestamp or the body will have been altered, or the public key will not verify the signature due to it not being signed by its private key.

Message signing also addresses a subtler class of attack: requests that are not replayed verbatim but modified before being relayed. An agent that intercepts a legitimate request, alters the parameters to access a different account or enlarge a transaction, and forwards the modified request will produce a signature that no longer matches the payload. The server rejects the request without ever needing to apply business logic. This shifts a meaningful class of authorization checks from application-layer logic, where mistakes are common, to the protocol layer, where the verification is uniform.

Why These Are No Longer Optional

Token Binding and Message Signing have historically been described as advanced or optional techniques—appropriate for high-value transactions but excessive for ordinary API traffic. That framing has not aged well. Agentic adversaries elevate the floor of what should be considered ordinary. An attacker with the capacity to run thousands of synthetic clients, capture tokens at scale, and orchestrate replay across them does not require a special motivation to do so; the abuse is essentially free at the margin. The defensive response must be to make replay structurally impossible rather than merely operationally unattractive.

©2026 Approov Limited | All rights reserved | Version 1.0

A Note on Implementation

Message Signing makes use of hardware-backed key storage on the device—the Secure Enclave on iOS, the StrongBox or TEE-backed Keystore on Android. Where this hardware is unavailable or has been compromised, the binding becomes weaker. This is one more reason that attestation is foundational: it is the mechanism by which the server learns whether the keystore on a given device can be trusted to hold a binding key in the first place.

4.3 Starving the Agent

Stepping back from the individual mechanisms, it is worth naming explicitly the strategy they collectively implement. The objective is not to detect agentic AI in flight, nor to outthink it after the fact. The objective is to deny it the materials it needs to function. An agent that has not extracted a key cannot impersonate. An agent that has not captured a long-lived token cannot replay. An agent that has not bypassed attestation cannot acquire a fresh token. An agent operating outside the bound context cannot use a token even if one is somehow obtained. Every one of these denials is structural rather than reactive, and every one of them scales without operational overhead.

The Zero Secrets Architecture, with token binding and message signing on top, is in this sense a starvation strategy. It turns the asymmetry that previously favored the attacker—cheap probing, expensive defense—back in the defender's favor. The attacker now has to solve hard cryptographic problems to gain anything, while the defender simply has to operate the architecture as designed.

©2026 Approov Limited | All rights reserved | Version 1.0

5. Adoption Roadmap

Most organizations cannot move from the current state to a fully Zero Secrets posture in a single quarter, nor should they try. The transition is best approached in stages, each of which delivers standalone value and creates a foundation for the next. The path described below is the one we have seen succeed across organizations of varying size, regulatory exposure, and engineering velocity.

Phase 1: Inventory and Baseline

The first task is to know what is actually in the application portfolio. Most teams underestimate the volume of secrets embedded in their mobile apps because the inventory has accumulated organically over years of integrations. A systematic scan of every published binary—both first-party and any white-label or partner builds—typically reveals a count higher than expected and a distribution wider than expected. The deliverable from Phase 1 is a single, accurate document that lists every secret currently present in every shipping artifact, classifies each one by category and sensitivity, and identifies the integration that requires it.

Alongside the inventory, baseline measurements should be taken on the API side. What is the volume of traffic against each API today? What proportion of it can be confidently attributed to genuine app instances? What is the cost—in compute, in third-party fees, in support load—of the traffic that cannot? These numbers establish the business case for the program and the metrics by which its success will be judged.

Phase 2: Establish Attestation

Attestation is the foundational pillar and should be deployed first. It can typically be added without modifying the existing API surface: the application begins requesting an attestation token, the gateway begins requiring it for sensitive endpoints, and unattested traffic is logged before it is blocked. The logging period is essential. It surfaces issues with edge devices, regional distribution, enterprise builds, and integration partners that would otherwise turn the eventual enforcement cutover into a support incident.

Once attestation is enforced, the immediate effect is the elimination of impersonation by repackaged or reverse-engineered clones. This alone is often enough to produce a measurable reduction in API abuse, before any of the secret-removal work begins.

Phase 3: Replace Static Pinning and Long-Lived Tokens

With attestation in place, two of the highest-leverage upgrades become straightforward. Static pinning is replaced with dynamic pinning delivered from the same control plane that handles attestation. Long-lived bearer tokens for API access are replaced with short-lived JWTs issued on the basis of valid attestation. Neither change requires modifying the third-party integrations that are the next phase's focus, and both deliver immediate operational benefits: faster certificate rotation, narrower exposure windows, easier audit.

©2026 Approov Limited | All rights reserved | Version 1.0

Phase 4: Remove Secrets

With the foundation in place, the inventory from Phase 1 becomes a work queue. Each entry is converted to either a proxy-pattern integration, in which the secret moves to a backend service that the app calls through, or a runtime-secrets integration, in which the secret is delivered from a backend store to attested instances on demand. The choice is per-integration: proxy where possible, runtime delivery where direct-to-SDK is unavoidable. The order is by sensitivity and exposure: the most damaging secrets first, the most widely exposed integrations next, and lower-sensitivity entries last.

This phase is the longest, but also the one whose progress is most legible to the business. Each removed secret is a discrete, demonstrable reduction in attack surface, and the inventory document from Phase 1 makes the burndown visible to executive leadership.

Phase 5: Protect the Token and the Request

With secrets removed and attestation enforcing client identity, the residual risk concentrates on token capture and request modification. Phase 5 closes those gaps by introducing token binding to associate JWTs with the app instances they were issued to, and message signing to bind individual requests to their content and timestamp. Either are additive changes that do not require revisiting earlier phases.

Phase 6: Operate, Measure, Iterate

The architecture is not a project but a posture. Once in place, it requires the ordinary attention any production control demands: dashboards on attestation pass and failure rates, alerting on anomalies, periodic review of the inventory to catch regressions during integration sprints, and quarterly rotation of the keys and certificates the system depends on. Mature programs also run scheduled adversarial exercises—internally or via specialist firms—that simulate agentic abuse against the live architecture and feed the results back into the design.

Sequencing Rule of Thumb

Attestation first, dynamic pinning and short-lived tokens second, secrets removal third, binding and signing fourth. Reversing this order tends to produce stalled programs and partial defenses. Following it tends to produce visible, measurable progress at every phase.

©2026 Approov Limited | All rights reserved | Version 1.0

6. The CISO Checklist

The following questions are intended to be asked of the team responsible for mobile and API security. They are not a maturity model in the formal sense; they are a practical instrument for separating programs that have adapted to the current threat environment from those that have not. A program that can answer all of them affirmatively is in good shape. A program that cannot answer any of them affirmatively is not yet engaged with the problem this paper describes.

Domain	Question	What a Strong Answer Looks Like
Inventory	Do we have a current, accurate list of every secret embedded in every published mobile artifact?	A maintained inventory updated each release; coverage of first-party and white-label builds; classification by sensitivity
Obfuscation	Are we relying on code obfuscation to hide secrets in the binary?	No primary reliance; obfuscation may be present as marginal friction but is not load-bearing
Pinning	Is certificate pinning enabled, and can pins be rotated without an app release?	Pinning is enforced and the pin set is delivered dynamically from a controlled service
Attestation	Do we cryptographically verify that API requests come from a genuine, unmodified app instance on an uncompromised device?	Cross-platform attestation is required for sensitive endpoints; failures are logged, alerted, and blocked
Secrets	For every third-party integration, is the credential delivered just-in-time to attested instances rather than embedded?	Proxy or runtime-secret patterns for all sensitive integrations; embedded credentials limited to non-sensitive cases with documented rationale
Tokens	Are API access tokens short-lived, scope-limited, and re-issued on the basis of fresh attestation?	JWTs with single-digit-minute lifetimes; scope claims enforced; re-issuance gated by current attestation state
Replay	Are tokens bound to the issuing app instance, and are sensitive requests signed?	Hardware-backed binding keys; per-request signatures covering method, path, body, and timestamp; server-side validation
Behavior	Are we relying on behavioral analytics as a primary control against automated abuse?	Behavioral signals are used for additional context only; primary defense is structural (attestation, binding, signing)

©2026 Approov Limited | All rights reserved | Version 1.0

Operations	Can we rotate any single key, certificate, or secret in our mobile architecture within minutes without an app release?	Yes, demonstrably, and rotation is exercised on a regular cadence
Adversarial	Have we tested the architecture against an agentic threat model in the past twelve months?	Yes, with documented findings, remediation, and re-test

©2026 Approov Limited | All rights reserved | Version 1.0

7. Conclusion

The mobile application has always been an awkward asset for the security team—distributed to environments outside the perimeter, executed on hardware outside enterprise control, updated on a cadence dictated by users rather than operators. For most of the last decade, the gap between this awkwardness and the value at stake was bridged by a set of techniques: obfuscation; embedded secret hiding; pinning; and behavioral analytics. Each layer imposed friction without providing structural protection. The bridge mostly held because the adversaries on the other side were also operating with friction.

That balance has shifted. Agentic AI removes the friction on the attacker's side. It compresses reverse engineering from days to minutes, fabricates plausible behavior at will, scales horizontally on cheap compute, and treats published binaries as inputs to its analysis rather than obstacles to it. Every defense that depended on imposing time or skill costs on a human attacker is now operating against an adversary for which time and skill are no longer scarce.

The Zero Secrets Architecture is the response that meets this adversary on its own terms. It does not try to be more clever; it simply removes the fuel the adversary depends upon. An agent that cannot find a key, cannot bind to an instance, cannot replay a token, and cannot synthesize a valid signature has nothing to attack, regardless of how much compute it commands or how sophisticated its planning becomes. Trust is anchored in dynamic pinning, in cryptographic attestation, in just-in-time secret delivery, and in short-lived bound tokens, none of which an adversary can synthesize from inspection of the binary.

For the CISO, the immediate question is no longer whether to adopt this approach but how quickly. The technical building blocks are mature. The integration patterns are well understood. The operational benefits—faster rotation, better visibility, lower support burden—are real and arrive early in the adoption curve. The cost of further delay is paid in the currency the agentic threat model favors most: a continuously growing attack surface, abused at machine speed, by adversaries that get cheaper, faster, and more capable with every passing quarter.

The organizations that move first will not have a perfect defense. They will have something better: an architecture in which the things an agent can extract are not the things that grant access. That structural property is what separates a security program that has adapted to the current environment from one that has not, and it is the foundation on which the next decade of mobile and API defense will be built.

About Approov

Approov Limited provides a unified mobile and API security platform that delivers cross-platform mobile app attestation, dynamic certificate pinning, runtime secrets protection, short-lived JWT issuance, and the supporting controls that together implement a Zero Secrets Architecture. Approov works across iOS, Android, and major cross-platform frameworks, and integrates with existing API gateways, identity providers, and secrets management systems. To learn how Approov can support your organization's adoption of a Zero Secrets posture, contact your Approov representative or visit the Approov website www.approov.com.

©2026 Approov Limited | All rights reserved | Version 1.0