NEXILIS SENTINEL

A STRATEGIC NOTE · 2026

The Key on the *Device*.

A short reflection for boards and senior executives on the largest quiet liability in Indonesian mobile banking — and the architectural answer that closes it.

PREPARED FOR

Directors, commissioners, and senior executives of regulated Indonesian institutions.

PT Easysoft Indonesia

NEXILIS.IO · JAKARTA

CONFIDENTIAL

Version 1.0

A NOTE TO THE READER

This is not a product brochure.

It is a short note, written for directors, commissioners, and senior executives of Indonesian institutions whose customers or constituents interact primarily through a mobile application. It describes a specific property of how those applications are protected today, the reason that property has quietly become the largest single security liability in Indonesian banking and public service, and what can be done about it.

It is organised as three short chapters. The first describes the situation as it actually is. The second describes three shifts that have recently made this situation urgent rather than theoretical. The third describes an architectural response — what an alternative construction of mobile application security looks like, and what it means for the institution that deploys it.

It is short on purpose. If after reading you would like to examine the technical detail, the regulatory mapping, or the commercial structure, those conversations are available on request.

— The authors

CHAPTER I

The situation as it *actually is*.

Begin with a familiar image. Tens of millions of Indonesians open a mobile banking application in the course of an ordinary Tuesday morning. They key in their PIN. They transfer money, pay bills, subscribe to a mutual fund, check the balance of a child's educational savings. The application, loaded on their device some months or years ago, verifies them, reaches the bank's servers, and executes the transaction.

None of this is remarkable. It is the infrastructure of ordinary Indonesian financial life, and it works, by and large, reliably.

What is worth examining — because it is not well understood outside the technical teams responsible for it — is what actually protects the application during those millions of daily interactions. The answer, uncomfortably, is that the application protects itself using a cryptographic key that lives on the user's device. Not on the bank's servers. Not in a secure facility. *On the user's phone*, alongside the encrypted application the key is designed to protect.

Every major commercial mobile app protection SDK in this category — DexGuard and iXGuard from Guardsquare, SHIELD from Promon, Arxan from Digital.ai, Zimperium's zShield and zKeyBox, Appdome, AppSealing, Verimatrix, Talsec, and the comparable products across the category — operates on this principle. The key is obfuscated. It is split across multiple locations in the binary. It is mathematically transformed through white-box cryptography. It may be buried under layers of considerable cleverness. But it is, nonetheless, derivable from material present on the device.

> *The protection key for your institution's most valuable customer-facing application exists, right now, on devices you do not own.*

And that means it is, in principle, extractable.

An attacker with sufficient time, a rooted device, and commercial reverse-engineering tools can — with effort but without breakthrough — extract that key. The tools required (Frida, Magisk, IDA Pro, Ghidra) are inexpensive or free, and widely documented. The skill is specialised but not rare; any competent mobile security researcher can perform this work, and increasingly, so can operators in fraud syndicates who have learned the technique from published research. The time required, for a protected banking application, is typically measured in weeks, not years.

And once the key is extracted, it is not extracted only for that one attacker, on that one device. It becomes the master key to every installation of that version of the application in circulation. It is sold. It is shared. It travels.

THE SITUATION, STATED PLAINLY

The protection key for your institution's most valuable customer-facing application exists, right now, on devices you do not own, in environments you cannot inspect, surrounded by tools whose sole function is to extract it.

This is not a theoretical vulnerability. It is not a zero-day awaiting disclosure. It is a structural property of how commercial mobile protection has been built for the last decade. It has been accepted because, until recently, the economics of extraction made it a slow and

expensive attack — a cost-raising defence rather than a prevention, but one sufficient for ordinary risk. That era is now ending, for reasons the next chapter describes.

CHAPTER II

The shift that has *already happened*.

Three developments, each ordinary in isolation, combine to make this an urgent rather than a theoretical problem. None is hypothetical. Each is visible in current threat intelligence, current regulatory publications, and current dark-web commercial activity observable to any institutional security team.

The first is the maturation of Android rooting ecosystems. Ten years ago, rooting an Android device required specialised knowledge and carried real risk to the device itself. Today, Magisk, Zygisk, and LSPosed are polished open-source tools with active user communities, tutorial videos in Bahasa Indonesia, and public forums on which the latest root-detection bypass techniques are shared within days of a banking app's release. The gap between the moment a security team deploys a new hardening measure and the moment that measure is publicly bypassed has narrowed, by most public accounts, from months to hours.

The second is the emergence of a mature economy for extracted application keys, reported by threat intelligence firms covering the region. Keys from banking applications circulate on Russian-language and Indonesian-language marketplaces, with prices varying substantially by target institution and the completeness of the associated documentation. The original extractor is rarely the final user. The key circulates, is resold through brokers, and eventually reaches fraud syndicates operating at industrial scale against the protected user base.

The third is the trajectory of Indonesian financial regulation. OJK POJK 11/2022 on digital banking resilience. POJK 29/2024 on consumer protection in financial services. SEOJK

provisions on mobile channel risk management. Bank Indonesia's SNAP framework and the associated SDK requirements. Read carefully — as compliance teams are currently reading them — each of these moves toward an expectation that financial institutions demonstrate device integrity verification, protection against instrumentation, and real-time response to compromised environments. These are expectations that conventional mobile protection SDKs architecturally cannot satisfy. The direction of regulatory travel is clear, even where specific requirements have not yet crystallised into enforceable obligations.

> *The gap between deploying a new hardening measure and having it publicly bypassed has narrowed, by most public accounts, from months to hours.*

Taken together, these three developments collapse a defensive posture that has held, approximately, for a decade. Mobile protection SDKs were designed for a threat environment in which key extraction was slow, rare, and performed by specialists. They are now operating against a threat environment in which extraction is commoditised, the resulting keys are fungible assets, and regulators are moving to require protections the existing architecture cannot provide.

And so the question worth putting to your chief information officer, your security committee, and your board risk committee is not whether your mobile application has been hardened. It almost certainly has. The right question is sharper than that.

THE QUESTION WORTH ASKING

What is our exposure the day a single extracted key compromises every installation at once?

For every institution currently relying on conventional mobile protection SDKs, this is not a hypothetical question. It is a scheduled one. The schedule is set by the attackers; the calendar is not shared.

CHAPTER III

The architectural *answer.*

WHAT FOLLOWS IS WORTH READING AS AN ARCHITECTURAL PROPOSITION rather than a product pitch. The question worth proposing is whether a different construction of mobile application protection is possible — one in which the uncomfortable property described in the first chapter simply does not exist.

The principle is a single sentence, and the rest of the architecture follows from it.

> *The decryption key for a protected application payload should never exist on a user's device without a live server authorisation that has verified hardware-rooted attestation of that specific device.*

In ordinary English: the application arrives on the user's phone as encrypted content only. The key to decrypt it is held on the institution's server infrastructure. At the moment the user launches the application, the phone presents cryptographic proof — signed by hardware the attacker cannot forge — that it is a genuine, unmodified device of the kind the institution expects. The server verifies this proof. If it holds, the server delivers a decryption key wrapped specifically to that device's hardware, usable only for that session. The key is used to load the application into memory, and is then immediately wiped.

At no point during or after this sequence does the decryption key rest on the user's device in a form that can be extracted. Static analysis of the application binary yields only ciphertext.

Runtime analysis sees a key that exists for the duration of a function call. An attacker with a rooted device, three months of time, and every commercial reverse-engineering tool in existence will not extract a key, because there is no key to extract.

Consider what this means in concrete operational terms. The difference between conventional protection and this architecture is best shown not as a specification comparison, but as two parallel timelines — what happens to an attacker working against a conventionally protected application, and what happens to the same attacker working against an application protected this way.

TIMELINE ONE

Under conventional protection.

A typical commercial mobile SDK.

WEEK 01	Attacker obtains a rooted Android device.
WEEK 02	Installs Frida, Magisk, and supporting reverse-engineering tools.
WEEKS 03–06	Systematic analysis of the application binary. Identifies where the protection key is stored and how it is decoded.
WEEK 08	Key extraction succeeds on the test device.
WEEK 10	Extraction verified against additional installations. Key is confirmed valid across the installed base.
WEEK 12	Key begins circulating on private marketplaces.
WEEK 16	Organised fraud operations use the key to clone the application and execute transactions against compromised user accounts.
WEEK 18	Institution detects the fraud pattern. Incident response begins.
WEEKS 19–20	Patched application prepared, submitted to Google Play and the App Store, approved, pushed to users. Customers who do not update remain exposed.

A substantial loss event, and a regulator conversation that will last considerably longer than the incident itself.

TIMELINE TWO

Under Sentinel.

Server-delivered keys, hardware attestation.

WEEK 01	Attacker obtains a rooted Android device.
WEEK 02	Installs Frida, Magisk, and supporting reverse-engineering tools.
WEEKS 03–06	Systematic analysis of the application binary. Discovers the application contains *no decryption key at any point during static analysis*.
WEEK 07	Observes the application at runtime. Sees a key briefly present during launch that is wiped within milliseconds, and that will not function on any other device.
WEEK 08	Attempts to replay the attestation token. Finds that each token is single-use and server-rejected on reuse.
WEEK 09	Attempts to run the application in an emulator. Hardware attestation fails cryptographically.
WEEK 10	Confirms there is no attack pattern against which patient offline analysis succeeds.

The timeline ends here. There is no extracted key. There is no installed-base compromise. There is no incident.

Illustrative timeline based on the general attack pattern observed across published mobile banking compromises. Specific durations vary by target, attacker skill, and toolchain.

The second timeline does not describe the elimination of all risk. Any serious architecture has residual risks, which we document in detail for technical evaluators. What this timeline describes is the elimination of the specific attack class — patient offline extraction of a master key that compromises the entire installed base — that is responsible for the large majority of consequential published mobile banking incidents. That single change re-orders what matters in the institution's risk posture.

From this architectural shift, three consequences follow. Each is relevant to a different part of the institution's risk posture, and each is worth considering on its own terms.

CONSEQUENCE I

Risk converts from unbounded to bounded.

The most common mobile banking incident under conventional protection is an extracted key that compromises the entire installed base. Under this architecture, that incident class simply does not occur. Individual incidents remain possible — any system has residual risk — but they are localised to specific devices, detectable in real time, and responsive to server-side intervention. The institution's risk profile shifts from *we will find out when we find out* to *we will know within minutes and can respond within minutes.*

CONSEQUENCE II

Regulatory posture moves ahead of the requirement.

The direction of Indonesian financial regulation is toward hardware attestation, device integrity verification, and demonstrable real-time response capability. This architecture was designed against those expectations rather than adapted to them. An institution that deploys it is, in effect, compliant with requirements that have not yet fully hardened — which means the next OJK or Bank Indonesia announcement does not trigger a disruptive SDK replacement programme eighteen months from now.

CONSEQUENCE III

Incident response collapses from weeks to minutes.

When a compromise is detected under conventional protection, the response loop is to identify the affected version, build a patched application, submit to Apple and Google, and wait seven to fourteen days for approval and user adoption. Under this architecture, the response is a server-side revocation that takes effect on the next application launch. The institution's window of exposure between detection and containment collapses from a fortnight to a few minutes. For a bank processing billions of rupiah an hour on mobile, that is not a technical improvement — it is the difference between a contained incident and a front-page crisis.

Where the conversation might *begin*.

If you have read this far, the architectural question has done its work. What remains is whether and how to examine it further.

What we would propose is a 90-minute executive briefing, conducted at your offices and attended by whichever of your senior leadership you would find it useful to include. The briefing covers the architecture in greater depth than this note permits, includes a live demonstration of the protection operating against instrumented attack scenarios, and concludes with a candid conversation about whether and how a formal evaluation might be structured. We will arrange it within two weeks of your request.

If you would prefer that the conversation begin at a technical level instead, we will provide the full product brief and the technical evaluation package under non-disclosure, and your chief information officer's team can begin the evaluation without executive involvement. Either path is available. The choice of which, and whether, is yours.

For planning purposes: a typical institutional deployment runs three to five months from signed engagement to general availability, with pre-contract evaluation typically adding two to four additional months. A serious evaluation of whether Sentinel is the right architectural fit can itself be completed in six to eight weeks.

PT Easysoft Indonesia

Jakarta, Indonesia · nexilis.io

sentinel@nexilis.io