International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(21s) 2026 pp. 1943-1955
https://doi.org/10.70917/ijcisim-2026-5970
Article

Architecting Resilient Asynchronous Identity Verification Platforms for Mobile Commerce: Integrating Secure SDK Lifecycles, State-Oriented Workflow Orchestration, Adaptive Polling, and Fault-Tolerant Recovery Mechanisms

Vamsi Krishna Minnakanti

Independent Researcher, USA
ORCID: 0009-0004-9758-5240

Abstract: Mobile commerce applications increasingly rely on identity verification to gate age-restricted purchases, financial actions, account recovery, and regulated transactions, yet the verification step itself rarely completes within a single request-response cycle. A user may finish document capture in seconds while the evaluating provider continues processing for minutes, and during that interval the mobile application must preserve context, tolerate network loss, and avoid submitting the same request twice. This article proposes an architecture that treats asynchronous identity verification as an explicit orchestration problem rather than a thin wrapper around a third-party software development kit. Four elements are integrated: a secure SDK lifecycle boundary that isolates provider-specific behavior behind an internal abstraction; a state-oriented workflow model that governs every transition a verification session can undergo; an adaptive polling and event-driven retrieval mechanism that balances responsiveness against network and server load; and a fault-tolerant recovery layer built on idempotent request handling and persisted workflow identifiers. Mobile lifecycle research and industry onboarding data both point toward the same underlying pressure: process termination and data loss are common rather than exceptional in production Android applications, deployment automation remains comparatively rare across studied codebases, and identity-verification abandonment has climbed as onboarding friction increases. What existing literature has not yet drawn together is how SDK isolation, explicit state modeling, adaptive retrieval, and idempotent recovery combine into one coherent verification-orchestration framework suited to mission-critical commerce and regulated healthcare contexts. This article develops that framework, positions it against mobile lifecycle management and distributed systems literature, and proposes an evaluation model spanning completion rate, recovery success, and duplicate-session frequency. The resulting architecture allows a mobile commerce platform to extend identity verification to account recovery, healthcare eligibility, and financial onboarding without binding the application to one provider's implementation details.
Keywords: Asynchronous Identity Verification, Mobile Commerce Security, State Machine Architecture, Adaptive Polling, Fault-Tolerant Recovery, SDK Lifecycle Management, Idempotent Transaction Processing

1. Introduction

A mobile commerce checkout that requires identity verification behaves differently from almost every other interaction in the application. Most mobile requests resolve in well under a second: a product loads, a cart updates, a search returns results. Identity verification breaks that pattern. A user completes document capture or a biometric scan, the mobile application submits that data to a third-party verification provider, and the provider's evaluation may

Copyright: © 2026 by the authors

continue for several seconds or several minutes after the on-device interaction has already ended. The person holding the phone may lock the screen, switch applications, lose cellular signal in an elevator, or simply wait, and the application must remain coherent through all of it.

Conventional API design does not anticipate this shape of problem. A synchronous request either succeeds or fails within a bounded window, and a client can treat the absence of a timely response as failure. Identity verification inverts that assumption: the absence of an immediate answer is the expected condition, not an error state, and the mobile application must stay useful, honest about progress, and safe against duplicate submission for as long as the underlying decision remains pending. Regulatory and fraud-prevention pressure has only made this scenario more frequent. Verification now gates age-restricted purchases, financial account actions, prescription and appointment access in healthcare settings, and delivery of high-value goods, so the asynchronous verification workflow has moved from a peripheral feature into a load-bearing part of the transaction path [1].

Engineering practice tends to treat this problem piecemeal even so. A team integrates a verification provider's SDK, wires its completion callback to a loading spinner, and adds a retry button for the case where the callback never returns. Under laboratory conditions this holds up reasonably well. Under real mobile lifecycle behavior it degrades quickly, because Android and iOS both reserve the right to terminate a backgrounded process to reclaim memory, and a workflow whose state lived only in memory does not survive that termination. Empirical analysis of Android applications has found lifecycle- and process-death-related data loss defects to be pervasive: one large-scale study identified 374 distinct data loss issues across a sample of only sixty-six applications, 284 of which had not been previously documented [13]. A verification workflow that has not been explicitly designed against this failure mode inherits the same fragility by default, not by exception.

This article argues that asynchronous identity verification should be architected as an explicit, state-governed orchestration problem spanning the mobile application, the verification provider's SDK, the commerce backend, and any downstream eligibility or compliance service. It should not be treated as an implementation detail hidden inside a single SDK integration. The proposed architecture rests on four coordinated mechanisms. A secure SDK lifecycle layer isolates the provider's specific API surface behind an internal boundary so that credential handling, token scope, and provider replacement remain the concern of one component rather than being scattered across the application. A state-oriented workflow model gives every verification session an explicit, enumerable set of states and transitions, replacing implicit assumptions about "waiting" and "done" with a structure that can be tested and audited directly. An adaptive polling and event-driven retrieval mechanism governs how the mobile application learns the verification outcome, adjusting its behavior to network quality, provider responsiveness, and application lifecycle state rather than polling at a fixed interval regardless of context. A fault-tolerant recovery layer, built on idempotency keys and persisted workflow identifiers, allows the system to resume, reconcile, or safely retry a verification session interrupted by network loss, application termination, or provider timeout.

Sections 3 through 6 develop each of these mechanisms in turn. Section 2 first situates the resulting architecture against related work in mobile lifecycle management, distributed idempotency, and identity-proofing standards, and Sections 7 through 10 extend the discussion into privacy and compliance obligations, user experience, evaluation, and broader applicability across regulated mobile workflows.

2. Related Work

Identity-proofing standards provide the regulatory and technical baseline against which any verification architecture must be measured. The National Institute of Standards and Technology's digital identity guidelines define identity assurance levels and the evidentiary requirements attached to each, establishing what a verification decision must actually demonstrate before a relying system can act on it [1], [6]. These guidelines describe the required rigor of the proofing process itself, but they leave the mobile-side orchestration of an asynchronous, multi-step proofing session largely unaddressed. That gap is precisely what this article's architecture is intended to close. Complementary mobile-specific guidance from the OWASP Mobile Application Security Verification Standard covers credential storage, transport security, and platform-level protections that a verification-adjacent mobile application should

satisfy, but it too stops short of prescribing how the application should model and recover the verification workflow itself [4].

Authorization standards for native applications inform the SDK lifecycle boundary proposed here. RFC 8252 establishes that native applications should treat authorization tokens as short-lived, scoped credentials obtained through a backend-mediated flow rather than embedded secrets, a principle this article extends to verification-session tokens more broadly [2], [3]. Subsequent security best-practice guidance formalizes attack patterns against native OAuth flows and the mitigations a compliant implementation is expected to apply, reinforcing that credential lifecycle management is an ongoing discipline rather than a one-time integration task [7]. A separate line of standards work addresses software development integrity in an adjacent but distinct way from the authentication-focused standards above. Where RFC 8252 and its associated security best-practice guidance govern how credentials move between a native application and an authorization server, the Secure Software Development Framework addresses how the software producing and consuming those credentials should itself be built, tested, and maintained over time [5]. The two concerns intersect at exactly the SDK boundary this article proposes: a credential-handling design can satisfy RFC 8252's token-scoping guidance perfectly and still be undermined by a development process that fails to verify, on an ongoing basis, that the boundary continues to enforce that scoping as the application evolves across releases. This article's SDK abstraction layer is best understood as the architectural mechanism that makes the framework's verifiability requirement checkable in practice, since a security review targeting one isolated component is considerably more tractable than a review that must trace credential handling across every call site where a provider's SDK might be invoked.

Mobile lifecycle management research documents the scale of the problem that an asynchronous verification workflow must withstand. Empirical studies of Android applications have catalogued data loss defects arising from configuration changes, process death, and navigation events, and have found that even mature, widely deployed applications retain unresolved instances of this defect class well after release [13]. A separate empirical study of continuous integration practice across 2,557 open-source Android projects found that only 9 percent incorporated deployment automation at all, and of the 238 projects that did deploy automatically, only 11 percent targeted the Google Play Store directly [14]. Read together, these two studies suggest that lifecycle-survival testing of the kind an asynchronous workflow requires is not yet standard practice across mobile release pipelines, which strengthens rather than weakens the case for building recovery discipline into the architecture itself instead of assuming it will be caught downstream.

Distributed-systems literature on idempotency and duplicate-request handling underpins the fault-tolerant recovery layer proposed in Section 6. The governing insight, that a request identifier allows a receiving system to recognize and safely absorb a retried submission without repeating its side effects, has become foundational to reliable payment and transaction architectures operating across unreliable networks [8]. Research applying machine learning techniques to e-commerce fraud detection reinforces why this matters beyond mere user convenience: duplicate and anomalous transaction patterns remain a persistent vector for financial loss in digital commerce, which means a verification-adjacent workflow cannot treat duplicate submission as only an interface annoyance [17].

Biometric liveness detection and facial verification research characterizes what the provider side of the verification exchange is actually capable of, which matters for how much confidence the SDK boundary should place in any single provider's output. Work applying deep learning architectures to biometric liveness detection has evaluated seven distinct network architectures against a spoofing dataset covering fourteen distinct attack techniques, from replay attacks to disguise-based spoofing [12]. Spectral-imaging approaches to live-face detection report an accuracy exceeding 98 percent across validation samples of real faces, printed photographs, and three-dimensional models, illustrating that provider-side reliability continues to be an active area of methodological improvement rather than a solved problem the mobile application can treat as uniform across vendors [11].

Adaptive network-interaction research supports the polling mechanism proposed in Section 5. Machine learning-based approaches to context-aware task scheduling on mobile devices have reported power consumption

reductions as high as 48 percent, alongside execution time reductions of up to 29 percent, when scheduling decisions account for network connection type and device context rather than proceeding on a fixed schedule regardless of conditions [15]. This finding transfers directly to the polling-interval decision this article addresses, since the underlying scheduling problem, when to act given uncertain network and server state, is structurally similar.

Regulated-industry deployment context is addressed from two directions in the literature. Research into mobile health application privacy and security practice has identified specific technical and procedural gaps between what mobile health developers implement and what patient data protection actually requires [18]. Recent work on consent-platform architecture for health data sharing demonstrates how granular, revocable consent can be structured for mobile-mediated data flows, organizing health information into distinct categories so that a user's consent for one category does not implicitly extend to others [16]. Industry measurement of onboarding abandonment quantifies the cost of getting verification friction wrong at scale: consumer surveys have recorded abandonment climbing from 40 percent in 2016 to 63 percent in 2020 and to 68 percent in 2022, with the average abandoning consumer giving up after roughly nineteen minutes [9]. Abandonment specifically attributable to the verification step has separately been identified as one of the most costly points of friction in the broader onboarding process, with roughly one in ten applicants abandoning financial-services onboarding on average [10].

3. Secure SDK Lifecycle Management

The mobile application should not treat a third-party identity-verification SDK as a component that sits directly inside the checkout or account-recovery flow. Doing so spreads provider-specific method calls, callback signatures, and error codes across the application, which turns a later provider replacement or supplementation into a substantial rewrite rather than a contained substitution. The SDK should instead sit behind an internal abstraction layer whose interface is defined by the application's own verification workflow, not by the provider's API surface. Every downstream component, including the state machine described in Section 4, interacts only with this internal boundary and never with the provider's SDK directly.

The lifecycle this boundary manages spans nine steps: requesting a verification session from the backend; receiving a short-lived, scoped SDK token; initializing the provider SDK with that token; launching document or biometric capture; receiving the SDK's local completion signal; persisting a workflow identifier that survives process death; retrieving the authoritative decision from the backend once the provider has finished server-side evaluation; applying that decision to the commerce workflow; and releasing SDK resources. Most architectures fail somewhere in the middle of this sequence, because the local SDK completion signal is frequently and incorrectly treated as equivalent to a finished verification. A successful local capture indicates only that identity data was captured and transmitted. It says nothing about whether the provider ultimately approved, rejected, or flagged the submission for manual review, and conflating the two is one of the more common design errors in verification integrations built without an explicit lifecycle boundary.

Table I. Verification Workflow States and Triggering Events

State	Triggering Event	Description
NotStarted	Application launch	No verification session exists yet.
SessionCreating	User initiates verification	Backend requests a short-lived, scoped SDK token.
CaptureReady	Token issued	Provider SDK initialized and ready for capture.
CaptureInProgress	Capture begins	User is actively completing document or biometric capture.
Submitted	SDK completion signal	Local capture finished; data transmitted to provider.

Processing	Backend acknowledgment	Provider is evaluating the submission server-side.
Approved	Provider decision: approved	Terminal state; commerce workflow proceeds.
Rejected	Provider decision: rejected	Terminal state; commerce workflow blocked.
RetryableFailure	Transient error (network, timeout)	Automatic retry may be attempted without user action.
NonRetryableFailure	Permanent error (unsupported document)	User must take corrective action to proceed.
TimedOut	Maximum wait duration exceeded	Terminal state; user notified of extended delay.
Cancelled	User cancellation (any state)	Terminal state; session explicitly abandoned by user.

Consider the concrete failure this boundary is designed to prevent. A verification SDK typically ships with a client-side initialization method that accepts an API key or client secret directly, and the fastest path to a working integration is to embed that credential in the application binary and call the SDK's initializer from wherever verification is first triggered in the code. This approach works in a demonstration and fails as a security control the moment the application is decompiled, because a credential embedded in application code is available to anyone who can extract the binary, regardless of how well the network transport around it is encrypted. The backend-mediated token exchange this section describes closes that gap by ensuring the only credential ever present on the device is a short-lived, single-purpose token scoped to one verification session, so that extracting it yields at most one already-time-limited session rather than standing access to the provider's account-level API.

Credential handling within this boundary follows directly from the native application OAuth guidance: the backend generates the verification session token, not the mobile application, and that token should be short-lived, scoped narrowly to the verification session it authorizes, and protected in transit and at rest through standard transport encryption and platform secure-storage mechanisms [2], [3], [7]. Embedding a long-lived or broadly scoped credential inside the application package reproduces exactly the native-application vulnerability class these standards were written to prevent. The risk here is not abstract. A verification-session token that outlives its intended window, or that carries more scope than the single verification action requires, becomes a durable target for extraction rather than a transient one, and the cost of that mistake compounds across every user session the token could be reused against.

Isolating the SDK behind this boundary also simplifies compliance review considerably. An organization operating in a regulated-healthcare context, where the mobile application handles both commerce transactions and health-eligibility verification, can subject the abstraction layer to a single security and privacy review rather than auditing every call site that directly invokes the provider's SDK. The Secure Software Development Framework emphasizes verifiable security properties throughout the development lifecycle, making it considerably easier to satisfy the requirement when the property in question, correct credential scoping and provider isolation, is concentrated in one component rather than distributed across dozens of call sites [5]. Concentrating the boundary this way also means that a security finding against the provider's SDK itself, rather than against the application's integration of it, can be addressed by updating one internal component instead of tracing every place the SDK's API was called directly.

4. State-Oriented Workflow Orchestration

The central architectural claim of this article is that asynchronous identity verification should be modeled as an explicit state machine, not as an implicit sequence of callback invocations. An implicit model, where application behavior is determined by whichever callback fired most recently, tends to produce conflicting dialogs, duplicate navigation events, and inconsistent interface states whenever two asynchronous signals arrive close together. A

provider timeout and a delayed backend response for the same session arriving within moments of each other are routine occurrences in production, not an edge case, and an implicit model has no principled way to decide which signal should win.

The proposed state machine defines eleven states: NotStarted, SessionCreating, CaptureReady, CaptureInProgress, Submitted, Processing, Approved, Rejected, RetryableFailure, NonRetryableFailure, TimedOut, and Cancelled. Each transition between states is triggered by a specific, named event: SDK completion, a backend response, a polling timeout, explicit user cancellation, or application restoration after process death. Because every transition is enumerable and testable in isolation, the workflow becomes deterministic in a way a callback-driven implementation is not. A given combination of current state and incoming event has exactly one defined outcome, which removes the entire class of bug where two near-simultaneous events leave the interface in an undefined condition that only a debugging session after the fact can untangle.

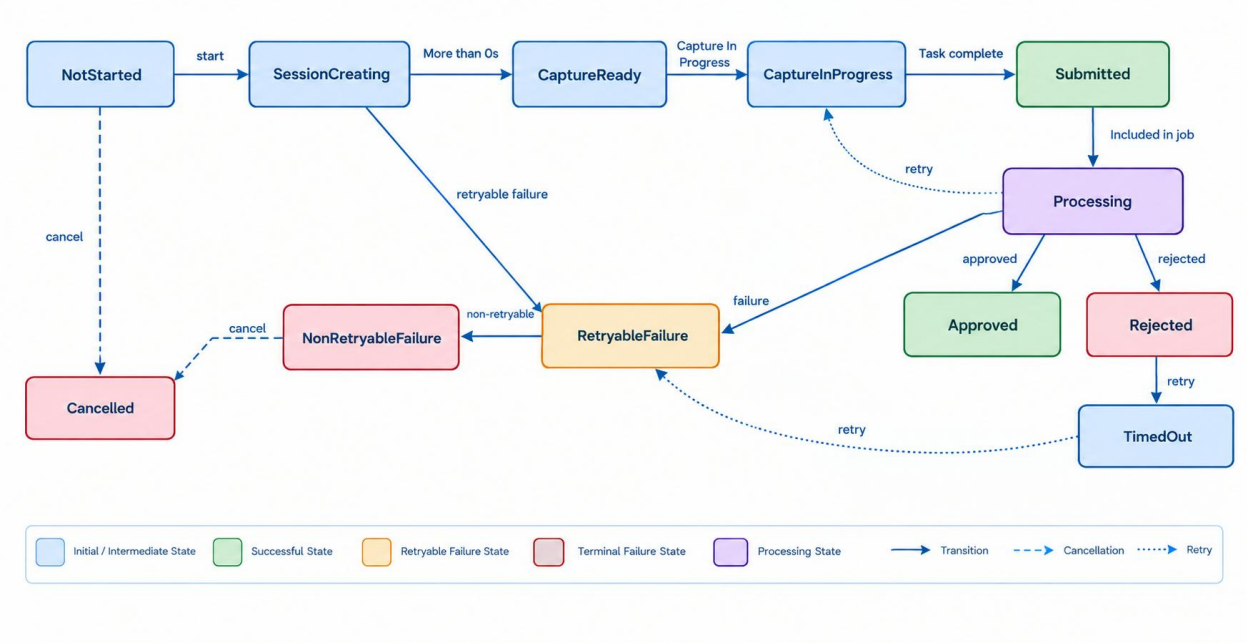


Fig. 1. State Machine Diagram for Asynchronous Identity Verification Workflow

The authoritative representation of this state should live in a lifecycle-aware component, such as a view model or dedicated workflow coordinator, rather than inside the interface elements that merely display it. This separation matters specifically because of the process-death behavior documented in mobile lifecycle research. If a verification state exists only as local variables inside a screen's view controller or activity, that state disappears the moment the operating system reclaims the process. Given how frequently data loss defects have been documented across even mature Android applications, treating this as a rare edge case understates the actual exposure [13]. A workflow coordinator that persists its state, at minimum the current status and the workflow identifier from Section 3, can restore the correct state after the application restarts, rather than forcing the user to begin verification again from the first step, which is exactly the outcome the abandonment statistics discussed in Section 2 warn against.

Distinguishing RetryableFailure from NonRetryableFailure as separate states, rather than collapsing them into a generic error condition, is what allows the recovery layer in Section 6 to behave correctly. A temporary network interruption belongs in RetryableFailure and can trigger an automatic retry without user involvement. An identity document that the provider cannot process, such as an unsupported document type, belongs in NonRetryableFailure and requires the user to take a different action, such as providing an alternate document. Treating these as the same state forces the interface to guess at the appropriate recovery behavior after the fact, which is precisely the ambiguity the state machine exists to remove in the first place.

5. Adaptive Polling and Result Retrieval

Once a verification session reaches the Submitted or Processing state, the mobile application must learn the eventual decision through one of two mechanisms: the application polls the backend for a status update, or the backend pushes that update to the application once the provider's decision arrives. Which mechanism is available depends on what the specific verification provider supports, and a production architecture should be designed to use both rather than committing to either one exclusively.

Controlled polling, where the application is the mechanism's only option, requires several safeguards beyond a naive fixed-interval loop. The client should observe an initial delay before the first status check, since immediate polling wastes a request cycle on a decision that has almost certainly not yet been reached. It should respect a configurable interval and a maximum wait duration, cancel outstanding polling if the user navigates away from the verification screen, and terminate immediately once a final decision, approved, rejected, or a non-retryable failure, arrives. Where the backend can supply retry-after guidance, the client should honor it rather than polling on its own fixed schedule, since the backend is typically better positioned to know how long the provider's processing window tends to run for that particular verification type.

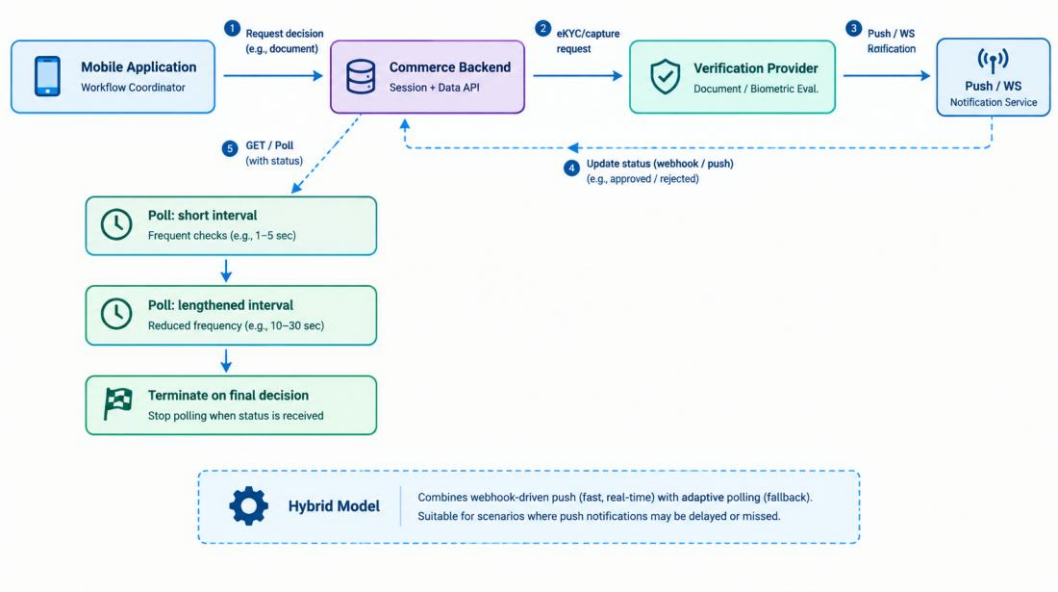


Fig. 2. Hybrid Webhook and Adaptive Polling Architecture

A fixed polling interval is nonetheless a poor fit for this problem, because provider processing duration, network quality, and server load all vary across sessions in ways a single interval cannot accommodate. An adaptive approach adjusts the polling interval based on the provider's current response pattern, the historical processing duration observed for similar verification types, current network quality, whether the application is in the foreground or background, and current server load. A practical implementation begins with a short interval immediately after submission, when a fast decision is still plausible, and lengthens that interval as the session remains pending, thereby reducing unnecessary request volume for sessions that take longer to resolve. Research on machine learning-based, context-aware task scheduling for mobile devices demonstrates that using network connection type and device context to adjust background behavior can reduce power consumption by up to 48 percent and completion time by up to 29 percent compared with context-blind scheduling [15]. That magnitude of improvement is directly relevant to the adaptive polling decision described here, since the underlying tradeoff, how aggressively to act under uncertain conditions, is the same one.

Where the verification provider supports it, an event-driven alternative avoids polling's inherent tradeoff between responsiveness and request volume altogether. The provider notifies the backend through a webhook once a

decision is reached, and the backend then pushes that update to the mobile client through a push notification, WebSocket connection, server-sent event stream, or a single lightweight retrieval request triggered by the push signal itself. A hybrid design, combining webhook-driven backend updates with client-side polling as a fallback for providers or network conditions that do not support the push path, captures the responsiveness benefit of the event-driven approach without depending on it being universally available across every provider the platform integrates.

6. Fault-Tolerant Recovery

An asynchronous verification workflow will encounter a predictable set of failure conditions in production: SDK initialization failure, camera or permission denial, document capture rejection by the provider, temporary network loss, provider-side timeout, backend failure, duplicate submission, application termination mid-session, expired verification sessions, and states where the provider and commerce backend disagree about the outcome. An architecture that has not planned explicitly for this list will encounter each condition eventually and improvise a response under production pressure rather than by design, which is a considerably worse position to be in than designing for the failure ahead of time.

Idempotency keys, generated when the verification session is first created and persisted alongside the workflow identifier described in Section 3, prevent two of the more damaging failure modes: duplicate verification sessions and repeated order-related side effects triggered by a retried request. When a client retries a submission after a network timeout, the backend can recognize the repeated idempotency key and return the original result rather than initiating a second, redundant verification attempt or a duplicate downstream action. This principle, that a request identifier allows a system to safely absorb a retried submission without repeating its effects, is well established in distributed systems and payment-processing practice, and it applies with equal force to identity-verification sessions, which share the same retry-under-uncertainty problem that payment requests do [8].

A concrete illustration clarifies how these mechanisms interact under an actual failure. Suppose a user completes document capture, the mobile application submits the verification request and transitions to Processing, and the operating system terminates the application process before the provider's decision arrives, perhaps because the user received a phone call and the system reclaimed memory during the interruption. When the user reopens the application, the workflow coordinator restores from the persisted state and finds a workflow identifier still in the Processing status. Rather than presenting the capture screen again, the correct behavior is to query the backend directly using that identifier. Three outcomes are possible at that point: the provider has already reached a decision, in which case the coordinator transitions immediately to Approved or Rejected without any further user action; the provider is still processing, in which case the coordinator resumes polling or waiting for the event-driven update exactly where it left off; or the session has expired past its validity window, in which case the coordinator transitions to NonRetryableFailure and the user is prompted to begin a new session. In none of these three branches does the user repeat document capture unnecessarily, and in none of them does the backend receive a duplicate verification request, because the idempotency key persisted alongside the workflow identifier ensures that even an accidental resubmission during this recovery sequence resolves to the original session rather than creating a second one.

The persisted workflow identifier also enables recovery after the application itself has been interrupted. If the process is terminated while a verification session is in the Processing state, restoring the application should retrieve the persisted identifier, query the backend for the session's current status, and transition the workflow coordinator directly to the correct state, rather than restarting the verification from the beginning. Given how frequently lifecycle-related data loss has been documented in production Android applications, an architecture that cannot perform this recovery step will force a meaningful share of users to repeat verification unnecessarily, which directly increases both processing costs and the abandonment risk already visible in onboarding-conversion research [9], [10], [13].

Recovery logic must also distinguish, at the point of failure, whether a given condition is retryable or requires user intervention, mirroring the RetryableFailure and NonRetryableFailure states from Section 4. A transient network interruption or a provider timeout under moderate load falls into the retryable category and can often be resolved automatically, without the user noticing that a retry occurred at all. A document rejected because it does not meet the

provider's format requirements, or a session that has genuinely expired past its validity window, falls into the non-retryable category and requires the user to take a corrective action, such as capturing a different document or restarting the session deliberately. Collapsing this distinction into a single generic retry policy either retries conditions that cannot succeed on retry, wasting time and creating false hope, or surfaces every failure to the user regardless of whether automatic recovery was actually available, creating unnecessary friction in either direction.

7. Privacy, Security, and Compliance

Identity verification workflows process some of the most sensitive data a mobile application will ever handle: government-issued document images, biometric samples, and the personal information those documents contain. The application's local footprint for this data should be minimized deliberately, rather than left to whatever the SDK integration happens to produce by default. Document images and biometric samples should remain within the verification provider's SDK sandbox or move directly to the backend, rather than being written to the application's general-purpose local storage or retained after the verification session concludes.

Beyond data minimization, the platform should implement encrypted transport for every network exchange involving verification data, use short-lived and narrowly scoped credentials as established in Section 3, provide secure on-device token storage using platform-provided secure storage mechanisms, redact sensitive fields from application logs, restrict analytics collection to non-identifying workflow events, establish explicit data-retention limits tied to regulatory requirements, present clear consent and disclosure language before capture begins, evaluate device integrity where the risk profile of the transaction warrants it, protect against screenshot capture or overlay attacks on verification screens, and maintain an auditable record of the policy decisions the workflow made at each stage.

In a regulated healthcare context, consent and disclosure requirements are particularly important, especially since the same mobile application may use this verification architecture for both commerce transactions and health-eligibility determinations. Research on mobile health application privacy practice has identified persistent gaps between developer implementation and the technical safeguards patient data protection actually requires, and these gaps do not close simply because the surrounding architecture is well engineered elsewhere [18]. Recent work on consent-platform architecture demonstrates a workable pattern for closing this gap: granular, revocable consent tied to specific categories of health-relevant data, structured so that a user's consent choice for one category does not implicitly extend to categories they did not agree to share [16]. A verification architecture operating across both commerce and healthcare contexts should apply this same granularity, rather than treating consent as a single blanket checkbox regardless of what the verification session will actually be used to authorize downstream [19].

8. User Experience and Accessibility

The interval during which a verification decision remains pending is an interface design problem as much as a backend one. Progress messaging during this period should communicate that information is being securely submitted and evaluated, without implying a certainty about the outcome or timeline the system cannot actually guarantee [20]. A message suggesting the decision will arrive within seconds, when provider processing can in practice extend to several minutes, creates a mismatch between stated and actual behavior [21]. That mismatch erodes user trust in the interface more thoroughly than an honestly uncertain message would, precisely because it sets an expectation the system then visibly fails to meet [22].

Indefinite loading indicators, unaccompanied by any explanation or time expectation, should be avoided in favor of interface states that communicate what is happening and what the user can do in the meantime, such as leaving the screen and receiving a notification when the result arrives. The interface should provide accessible text for all verification states, screen-reader-compatible status announcements, predictable back-navigation behavior that does not silently discard a pending session, and a clear recovery path when a failure does occur [23]. A non-dismissible dialog may be appropriate during one specific critical instant, the moment a capture is actively being transmitted and interrupting it would corrupt the submission, but such cases should be a narrow, deliberately bounded exception rather

than the default behavior for the entire verification interval. The user should never be left in a state from which there is no eventual path forward, regardless of how the underlying verification session resolves [24].

9. Evaluation Framework

Assessing whether an asynchronous identity-verification architecture performs as intended requires measurement across several dimensions, since a single metric such as raw completion rate can mask problems that only appear when examined from a different angle. Verification completion rate establishes the baseline: what proportion of initiated sessions reach a final decision at all, as opposed to abandonment or indefinite pending status? Average and percentile processing duration, particularly the higher percentiles rather than only the mean, reveal how the architecture performs for the sessions that take longest, which are also the sessions most exposed to lifecycle interruption and network instability.

Duplicate-session frequency measures whether the idempotency mechanisms described in Section 6 are functioning as intended. A nonzero rate here indicates either a defect in idempotency-key handling or a retry policy more aggressive than the underlying failure conditions actually warrant. Polling request volume, tracked against the adaptive polling mechanism in Section 5, indicates whether the interval-adjustment logic is genuinely reducing unnecessary network traffic compared with a fixed-interval baseline. Recovery rate after application interruption directly measures the fault-tolerant recovery layer's real-world effectiveness, and timeout rate indicates whether the maximum wait duration and retry-after guidance are calibrated appropriately for the provider's actual processing behavior.

Each of these metrics is more useful when defined operationally rather than left as an abstract label. Verification completion rate should be computed as the proportion of sessions that reach Approved, Rejected, or a terminal NonRetryableFailure state within a defined observation window, typically forty-eight hours, rather than left open-ended, since an open-ended window makes the metric impossible to report on a regular cadence. Recovery rate after application interruption should specifically isolate sessions that were in Processing or Submitted at the moment of process termination and measure what fraction of those sessions reach a terminal state without requiring the user to recapture identity documents, since this is the specific behavior Section 6's recovery layer is meant to guarantee. Duplicate-session frequency is best expressed as the count of sessions sharing an idempotency key that nonetheless produced more than one provider-side verification attempt, divided by total session count, since the metric isolates genuine idempotency failures from the far more common and benign case of a user simply retrying a session that failed for a legitimate, non-retryable reason.

Table II. Evaluation Metrics for Asynchronous Identity Verification Architectures

Metric	Operational Definition	Target Range
Verification completion rate	Proportion of sessions reaching Approved, Rejected, or terminal NonRetryableFailure within a 48-hour observation window.	Above 90%
Recovery rate after interruption	Fraction of sessions in Processing/Submitted at process termination that reach a terminal state without recapture.	Above 95%
Duplicate-session frequency	The duplicate-session frequency is the number of sessions sharing an idempotency key that produced more than one provider-side attempt, divided by the total number of sessions.	Below 0.1%
Polling request volume	Average status-check requests per session under adaptive interval versus fixed-interval baseline.	30-50% reduction vs. baseline
Timeout rate	Proportion of sessions that reach TimedOut before a final decision arrives.	Below 2%

Checkout abandonment post-verification	Proportion of approved sessions that do not complete the associated commerce transaction.	Tracked; no fixed target

Provider-specific failure distribution allows an organization operating with more than one verification provider to identify whether a given failure pattern is a property of the architecture or a property of one provider's implementation, which matters directly for the provider-replacement flexibility the SDK abstraction layer in Section 3 is designed to preserve. Checkout abandonment following verification connects this architecture's performance to commerce outcomes directly, since a verification step that functions correctly in isolation but still drives users away at a high rate has not actually solved the underlying business problem it was built to address. Security and privacy violation counts, ideally zero, and accessibility completion results across users relying on assistive technology round out an evaluation framework that treats reliability, security, and usability as equally necessary rather than treating any single dimension as sufficient on its own.

10. Broader Implications

An architecture built around the four mechanisms described in this article, SDK lifecycle isolation, explicit state-oriented orchestration, adaptive result retrieval, and idempotent fault-tolerant recovery, generalizes beyond the mobile commerce checkout scenario that motivates it. The same orchestration framework applies wherever a mobile application must coordinate an asynchronous, multi-system decision process with real consequences for the user: account recovery flows that require re-verifying identity before restoring access, healthcare eligibility determinations that gate appointment scheduling or prescription access, financial account onboarding subject to the same identity-assurance requirements as an initial purchase, delivery of regulated or age-restricted goods, and employee or contractor identity verification in operational contexts [22].

Because the SDK abstraction layer isolates provider-specific behavior from the rest of the application, an organization is not permanently bound to a single verification vendor's implementation choices. Provider replacement, multi-provider routing for cost or reliability reasons, or extending the same orchestration framework to an entirely different verification use case become integration exercises within a stable internal boundary, rather than each requiring a rewrite of application-wide logic tied to one vendor's specific SDK behavior. This flexibility becomes more valuable, not less, as an organization's verification needs expand across multiple regulated domains that may each favor a different specialized provider [24].

11. Conclusion

Asynchronous identity verification is a distributed workflow that a mobile application initiates. It is not a single request-response exchange that a callback can adequately represent, and treating it as the latter produces architectures that function acceptably in a controlled demonstration and degrade under the real conditions of mobile networks, operating-system process management, and provider processing variability. This article has proposed an architecture that instead treats verification as an explicit orchestration problem: a secure SDK lifecycle boundary that isolates provider-specific behavior, a state-oriented workflow model that makes every transition enumerable and testable, an adaptive polling and an event-driven retrieval mechanism calibrated to network and provider conditions, and a fault-tolerant recovery layer built on idempotency and persisted workflow state.

Evidence from mobile lifecycle research, distributed-systems literature, and industry onboarding data converges on the same conclusion from different directions. Process death and lifecycle interruption are common rather than exceptional. Duplicate-request handling is a well-established but frequently under-implemented discipline. Verification friction carries a measurable, worsening cost in abandonment. An architecture that treats identity verification as a first-class orchestration capability, rather than an incidental SDK integration, is better positioned to absorb these realities while extending naturally to the broader set of regulated and mission-critical mobile workflows that increasingly depend on the same underlying verification problem.

Ethics Statement

Conflict of Interest: The author declares no conflict of interest.

Statement of Human and Animal Rights: This study did not involve human participants, animal subjects, or identifiable personal data requiring ethics board review. The architecture discussed is conceptual and analytical in nature.

Informed Consent: Not applicable.

CRediT Author Contribution Statement

Vamsi Krishna Minnakanti: Conceptualization, Methodology, Writing - Original Draft, Writing - Review & Editing, Visualization.

REFERENCES

1. National Institute of Standards and Technology, "Digital Identity Guidelines (SP 800-63-4)," NIST, 2025. Available: https://pages.nist.gov/800-63-4/sp800-63.html
2. Internet Engineering Task Force, "The OAuth 2.0 Authorization Framework (RFC 6749)," IETF, 2012. Available: https://www.rfc-editor.org/rfc/rfc6749.html
3. W. Denniss and J. Bradley, "OAuth 2.0 for Native Apps (RFC 8252)," Internet Engineering Task Force, 2017. Available: https://www.rfc-editor.org/rfc/rfc8252.html
4. OWASP Foundation, "Mobile Application Security Verification Standard (MASVS) v2.1.0," OWASP, 2024. Available: https://mas.owasp.org/MASVS/
5. National Institute of Standards and Technology, "Secure Software Development Framework (SP 800-218)," NIST, 2022. Available: https://csrc.nist.gov/pubs/sp/800/218/final
6. P. A. Grassi, J. L. Fenton, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene and M. F. Theofanos, "SP 800-63A: Digital Identity Guidelines - Identity Proofing and Enrollment," NIST, 2017. Available: https://pages.nist.gov/800-63-3/sp800-63a.html
7. T. Lodderstedt, J. Bradley, A. Labunets and D. Fett, "OAuth 2.0 Security Best Current Practice (RFC 9700)," Internet Engineering Task Force, 2025. Available: https://datatracker.ietf.org/doc/html/rfc9700
8. A. Ghadge, "Why Idempotency Matters in Payment Processing Architectures," IEEE Computer Society Tech News, 2024. Available: https://www.computer.org/publications/tech-news/trends/idempotency-in-payment-processing-architecture
9. Signicat, "The Battle to Onboard 2022," Signicat, 2022. Available: https://www.signicat.com/the-battle-to-onboard-2022
10. OneID, "Why 68% of Users Abandon Identity Checks (And What It Costs Your Business)," OneID, 2023. Available: https://oneid.uk/news-and-events/identity-verification-abandonment-cost
11. Z. Wang, S. Wang, W. Yu, B. Gao, C. Li and T. Wang, "Accurate Real-Time Live Face Detection Using Snapshot Spectral Imaging Method," Sensors, 2025. Available: https://doi.org/10.3390/s25030952
12. S. Khairnar, S. Gite, K. Mahajan, B. Pradhan, A. Alamri and S. D. Thepade, "Advanced Techniques for Biometric Authentication: Leveraging Deep Learning and Explainable AI," IEEE Access, 2024. Available: https://doi.org/10.1109/ACCESS.2024.3474690
13. W. Guo, Z. Dong, L. Shen, W. Tian, T. Su and X. Peng, "Detecting and Fixing Data Loss Issues in Android Apps," Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2022. Available: https://doi.org/10.1145/3533767.3534402
14. T. A. Ghaleb, O. Abduljalil and S. Hassan, "CI/CD Configuration Practices in Open-Source Android Apps: An Empirical Study," ACM Transactions on Software Engineering and Methodology, 2025. Available: https://doi.org/10.1145/3736758
15. P. Nawrocki and B. Sniezynski, "Adaptive Context-Aware Energy Optimization for Services on Mobile Devices with Use of Machine Learning," Wireless Personal Communications, 2020. Available: https://doi.org/10.1007/s11277-020-07657-9
16. S. Bruckner, A. Dridi, A. Deshmukh, T. Kirsten, A. Lauber-Ronsberg, R. Riedel, S. Hetmank, C. Welzel and S. Gilbert, "A User-Driven Consent Platform for Health Data Sharing in Digital Health Applications," npj Digital Medicine, 2025. Available: https://doi.org/10.1038/s41746-025-02147-3
17. A. Mutemi and F. Bacao, "E-Commerce Fraud Detection Based on Machine Learning Techniques," Big Data Mining and Analytics, 2024. Available: https://doi.org/10.26599/BDMA.2023.9020023
18. R. Rezaee, M. Khashayar, S. Saeedinezhad, M. Nasiri and S. Zare, "Critical Criteria and Countermeasures for Mobile Health Developers to Ensure Mobile Health Privacy and Security: Mixed Methods Study," JMIR mHealth and uHealth, 2023. Available: https://doi.org/10.2196/39055
19. A. Bhatnagar and V. R. K. Bonagiri, "Operational Excellence through Cloud-Native Distributed Systems," *International Journal of Computational and Experimental Science and Engineering*, vol. 12, no. 2, 2026. Available: https://doi.org/10.22399/ijcesen.5207

20. K. Rekulapalli, S. R. Kagga, V. Ayyagari, A. Akula, and R. A. Raj Akula, "AI in HRM: Enhancing Workforce Competency and Organizational Effectiveness," *SSRN*, 2026. Available: https://ssrn.com/abstract=6762998
21. K. Rekulapalli, H. Nilagiri, and M. Vakkanthula, "A comparative evaluation of national FHIR Implementation Guides: Strategic methodologies for enhancing interoperability, regulatory alignment, and clinical comprehensiveness in global health IT," *Health Systems*, pp. 1–13, 2026.
22. R. K. Shaw, "Kerberos vs. Public Key Authentication in z/OS Environments: Performance, Scalability, and Security Trade-offs," in *2026 9th International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE, pp. 1305–1312, 2026.
23. H. Gaggar, "MemGov: Policy-Governed Memory Lifecycle Management for Enterprise LLM Agents," *Journal of Intelligent Decision Making and Information Science*, vol. 3, no. 11s, pp. 2199–2214, 2026.
24. H. Gaggar, "Architecting scalable enterprise AI agent infrastructure for distributed intelligent systems," *International Journal of Advances in Signal and Image Sciences*, pp. 193–208, 2025. Available: https://doi.org/10.29284/da9w0117