

TECHNICAL RESEARCH PAPER  
*Enterprise Mobility and Device Management*

---

# ANDROID DEVICE ARCHITECTURE AND MOBILE DEVICE MANAGEMENT IN THE QUICK SERVICE RESTAURANT INDUSTRY

*Hardware Specifications, Software Architectures, Deployment Patterns, and  
Real-World Application of Enterprise Android Platforms*

---

**Shalkhar Namyrkozhoev**

Technical Consultant, Android Device Management  
Esper, Inc. Bellevue, Washington, USA  
Master of Communication in Communities and Networks Candidate  
University of Washington Seattle, Washington, USA

---

Research Period: November 2025 May 2026

Published: May 2026

## Abstract

This paper presents a technical analysis of Android device hardware, software architecture, and mobile device management (MDM) implementation in the Quick Service Restaurant industry. The analysis is grounded in six months of hands-on technical consulting work at Esper, Inc. The research was conducted from November 2025 through May 2026 and draws on device-level assessments, MDM platform evaluations, and deployment architecture reviews across QSR operators managing fleets ranging from several hundred to several thousand Android endpoints.

The QSR industry operates one of the most hardware-diverse, performance-constrained, and operationally demanding Android deployments in the commercial sector. Self-order kiosks, kitchen display systems, point-of-sale terminals, and staff-facing handhelds each impose distinct hardware requirements, OS configuration constraints, application stack demands, and MDM policy architectures. Understanding these devices at the level of processor architecture, memory configuration, OS version, and management API capability is a prerequisite for designing deployments that remain stable, secure, and maintainable at scale.

The paper examines the hardware specifications of the dominant commercial Android devices in QSR deployments, including the PAX A920 Pro POS terminal, the Sunmi K2 self-order kiosk, the Sunmi T2 countertop POS, the Samsung QMR-B commercial display, and the Zebra TC53 rugged handheld. It then analyzes the Android Enterprise API architecture that governs how these devices are enrolled, locked down, updated, and remotely managed, with specific attention to the AMAPI dedicated device policy model, OEMConfig extensions, zero-touch enrollment, and the kiosk mode implementation at the OS level. The paper concludes with a technical framework for MDM platform selection and deployment architecture design in QSR contexts.

---

## Table of Contents

1. Background and Technical Motivation
2. Research Questions and Technical Hypotheses
3. Android in the Commercial Device Ecosystem
4. Research Scope and Methodology
5. Hardware Architecture of QSR Android Devices
6. Android Enterprise Software Architecture
7. MDM Policy Implementation and Kiosk Mode
8. OEM-Specific Management Extensions
9. Real-World Deployment Observations
10. OS Fragmentation and Security Patch Analysis
11. Emerging Technical Developments

- 12. Technical Framework for MDM Selection
- 13. Conclusions
- 14. Acknowledgments
- 15. References

# 1. Background and Technical Motivation

## 1.1 The QSR Android Fleet as a Computer Science Problem

When this research began in November 2025, one of the first patterns to become clear was how consistently QSR operators described their device management problems in operational language while the underlying causes were almost always technical. A brand describing a problem as 'kiosks going down during the lunch rush' was describing an application crash that the OS-level kiosk lock did not recover from automatically because the MDM policy had set `installType` to `FORCE_INSTALLED` rather than `KIOSK` in the AMAPI policy JSON. A brand describing 'update problems across locations' was describing an OTA update rollout that had no staged deployment configuration and no rollback policy, resulting in application compatibility failures on devices that had not been updated to the prerequisite API level.

These are computer science problems. They involve Android API usage, process management, package installation architecture, OS version constraints, and network-layer policy delivery. They require technical analysis to diagnose and technical solutions to resolve. The gap that motivated this paper is that the available literature on Android MDM in QSR contexts either abstracts away the technical layer entirely, treating MDM as a procurement decision rather than a systems architecture problem, or addresses technical MDM topics without situating them in the operational realities of the QSR environment where the devices actually live.

## 1.2 The Technical Consultant Perspective

In the role of Technical Consultant at Esper, the work over the research period encompassed device onboarding architecture reviews, AMAPI policy design consultations, OEMConfig configuration for Samsung Knox and Sunmi devices, zero-touch enrollment implementation planning, and OTA update strategy design for multi-location QSR deployments. This work provided access to the technical layer of device management that is rarely documented in practitioner literature: the specific API calls, policy configurations, enrollment methods, and hardware constraints that determine whether a fleet of several thousand kiosks remains in service.

This paper documents what I observed and learned in that role, organized as a technical analysis that can be used by computer science practitioners, MDM platform engineers, and QSR technology architects to make better decisions about Android device deployment and management at scale.

## 2. Research Questions and Technical Hypotheses

### 2.1 Research Questions

- RQ1: What are the hardware specifications, processor architectures, OS versions, and management API capabilities of the dominant commercial Android devices in QSR deployments, and how do these characteristics constrain MDM implementation choices?
- RQ2: How does the Android Enterprise dedicated device API model, implemented through AMAPI, translate into the specific policy configurations required for stable, secure QSR kiosk and POS terminal deployments?
- RQ3: What OEM-specific management extensions, including Samsung Knox OEMConfig and Sunmi device management APIs, add or constrain management capability beyond the base Android Enterprise framework?
- RQ4: What OS fragmentation exists across active QSR Android deployments, and what are the specific security patch and API level implications of that fragmentation for MDM policy design and PCI DSS v4.0.1 compliance?
- RQ5: What technical criteria should drive MDM platform selection for QSR dedicated-device deployments, and how do leading platforms differ at the implementation level?

### 2.2 Technical Hypotheses

- H1: The primary source of MDM underperformance in QSR dedicated-device deployments is incorrect use of the Android Enterprise application installType field. Operators whose MDM platforms set installType to FORCE\_INSTALLED rather than KIOSK for their primary kiosk application experience significantly higher crash-recovery failure rates because FORCE\_INSTALLED does not engage the OS-level lock task mode that KIOSK installType activates.
- H2: OEM-specific management extensions, particularly Samsung Knox OEMConfig and the Knox E-FOTA firmware-over-the-air update system, provide materially superior OS update control compared to the base AMAPI systemUpdate policy for operators with Samsung-dominant hardware fleets, because they allow update targeting at the specific firmware build version rather than only the Android version.
- H3: The concentration of QSR Android deployments on API levels below 30 (Android 11) significantly limits the MDM policy capabilities available to operators because key dedicated-device APIs introduced in Android 11 and 12, including enhanced lock task mode controls and the DevicePolicyManager getEnrollmentSpecificId method, are unavailable on older OS versions.

## 3. Android in the Commercial Device Ecosystem

### 3.1 AOSP, GMS, and Commercial Android Variants

The Android operating system that powers QSR commercial devices exists in several technically distinct variants that have significant implications for MDM implementation. The Android Open Source Project (AOSP) is the base platform. Google Mobile Services (GMS) is the suite of Google-proprietary applications and APIs, including Google Play, the Play Protect security framework, and the Android Management API client (Android Device Policy), that OEMs license from Google and layer on top of AOSP. Most commercial Android devices in QSR deployments run GMS-certified builds, which is a prerequisite for AMAPI-based management.

A third variant is relevant to QSR deployments: PayDroid, the PAX Technology-developed Android extension used on PAX payment terminals. PayDroid layers a payment-hardened security framework on top of AOSP, adding hardware security module (HSM) integration, payment application sandboxing, and PCI PTS-compliant key injection infrastructure. PayDroid devices support a subset of Android Enterprise management capabilities but do not support the full AMAPI feature set, which has practical implications for MDM policy design discussed in Section 7.

Zebra Technologies takes a third approach with its StageNow provisioning framework, which supports device enrollment and configuration management on both GMS and non-GMS (AOSP) Zebra devices. StageNow uses Zebra's MX (Mobility Extensions) framework, which provides OEM-specific configuration capabilities comparable in scope to Samsung Knox OEMConfig but implemented through a proprietary XML-based profile system rather than the standard OEMConfig managed configuration architecture.

### 3.2 The Android Enterprise API Stack

The Android Enterprise API stack, as documented in the Android developer documentation updated through March 2026, consists of four layers that are relevant to QSR MDM implementation. At the hardware layer, the Trusted Execution Environment (TEE), implemented as ARM TrustZone on virtually all commercial Android SoCs, provides the hardware-backed key storage and cryptographic attestation that underpins device identity verification and the Android Keystore system. At the OS layer, the DevicePolicyManager (DPM) API exposes the device management capabilities that MDM platforms invoke through a Device Policy Controller (DPC) application installed on the managed device.

The third layer is the Android Management API (AMAPI), Google's cloud-side management infrastructure that allows MDM platforms to deliver policies to enrolled devices through the Android Device Policy DPC application without building and maintaining their own DPC. AMAPI communicates with enrolled devices through Firebase Cloud Messaging (FCM) for

real-time policy push and uses a polling fallback for environments where FCM is blocked. The fourth layer is OEMConfig, the standardized managed configuration architecture that OEMs use to expose hardware-specific management capabilities to any AMAPI-compatible MDM platform through a declarative JSON schema.

## 4. Research Scope and Methodology

### 4.1 Technical Evidence Sources

The methodology for this paper combines four technical evidence streams. First, direct hands-on technical consulting work at Esper from November 2025 through May 2026, encompassing AMAPI policy design, OEMConfig configuration, zero-touch enrollment implementation, and OTA update strategy design for QSR operators. Second, systematic review of primary technical documentation: the Google Android Enterprise developer documentation and AMAPI reference updated through April 2026, the Samsung Knox documentation at docs.samsungknox.com, the Sunmi device management documentation, the Zebra StageNow and OEMConfig technical guides, and the PAX Technology PAXSTORE MDM integration documentation. Third, device-level hardware specification analysis drawn from OEM published specification sheets and independent hardware teardowns. Fourth, OS version and security patch distribution analysis based on the author's observations across Esper-managed QSR device fleets during the research period.

### 4.2 Device Selection Rationale

The five device families selected for detailed hardware analysis represent the dominant platforms encountered in QSR Android deployments during the research period. PAX A920 Pro POS terminals were present in the majority of QSR payment terminal deployments reviewed. Sunmi K2 self-order kiosks and Sunmi T2 countertop POS units were the most commonly observed non-Samsung kiosk and POS hardware. Samsung QMR-B commercial displays were the dominant platform for digital menu boards and drive-through order confirmation displays. Zebra TC53 rugged handhelds were the primary device for staff-facing inventory, order taking, and operational applications in full-service QSR formats.



## 5. Hardware Architecture of QSR Android Devices

### 5.1 PAX A920 Pro: Android Payment Terminal

The PAX A920 Pro is the most widely deployed Android payment terminal in North American QSR as of 2026. It runs the PAX PayDroid platform, which is built on Android with payment-specific hardening. The A920 Pro is powered by a Qualcomm Cortex A75 and A55 multi-core processor, supports 4G LTE and WiFi 802.11 a/b/g/n/ac connectivity, and includes an integrated thermal receipt printer, a 5.99-inch HD touchscreen, EMV chip, NFC contactless, and magnetic stripe card readers. The device is PCI PTS 6.x certified and supports PIN entry through a hardware-encrypted PIN pad.

From an MDM architecture perspective, PAX A920 Pro devices managed through the PAXSTORE platform support remote application deployment and update, device monitoring, and basic policy enforcement. However, full AMAPI enrollment is not supported on PayDroid devices because the PayDroid security architecture restricts the device owner provisioning path required for Android Enterprise Fully Managed or Dedicated Device enrollment. QSR operators managing PAX terminals through Esper or other standard AMAPI-based MDMs typically use an integration approach that bridges PAXSTORE's application management with the broader MDM fleet's monitoring and reporting capabilities.

Specification	PAX A920 Pro
Processor	Qualcomm multi-core (Cortex A75 + A55)
Operating System	Android 14 with PayDroid security layer
Display	5.99-inch HD touchscreen
Connectivity	4G LTE, WiFi 802.11 a/b/g/n/ac, Bluetooth 5.0
Payment Interfaces	EMV chip, NFC contactless, magnetic stripe
Security Certification	PCI PTS 6.x, PCI DSS compliant hardware
MDM Enrollment	PAXSTORE native; limited AMAPI support
Printer	Integrated 2-inch thermal receipt printer
Camera	Rear-facing camera with QR code scanning

### 5.2 Sunmi K2: Self-Order Kiosk

The Sunmi K2 is a purpose-built self-order kiosk running full Android, making it one of the most MDM-friendly kiosk platforms in the QSR hardware ecosystem because it supports standard Android Enterprise Fully Managed and Dedicated Device enrollment through AMAPI. The K2 features a large vertical touchscreen display, integrated receipt printer, barcode scanner,

and support for card and contactless payments. Unlike PAX PayDroid terminals, Sunmi devices run Google-certified Android with GMS, enabling full AMAPI policy enforcement including the KIOSK installType that activates OS-level lock task mode.

Sunmi provides an OEMConfig application that exposes Sunmi-specific management capabilities through the standard managed configuration framework. This includes control over the Sunmi system launcher, the SunmiPay payment integration layer, hardware peripheral configuration (scanner sensitivity, printer settings, display brightness scheduling), and the SunmiOS update channel selection. The OEMConfig integration means these hardware-specific settings can be managed through any AMAPI-compatible MDM platform without requiring Sunmi-proprietary management tools.

Specification	Sunmi K2
Operating System	Android (GMS certified, full AMAPI support)
Display	21.5-inch vertical touchscreen
Management	Full AMAPI enrollment, OEMConfig extensions, Sunmi MDM
Kiosk Mode	KIOSK installType supported, OS-level lock task mode
Payment	Integrated NFC, EMV, QR code scanner
Printer	Integrated 80mm thermal printer
OEMConfig	Sunmi OEMConfig app for hardware-specific management
Deployment	QR code, NFC, or zero-touch enrollment

### 5.3 Sunmi T2 Mini: Countertop POS

The Sunmi T2 Mini is an Android-based countertop POS system widely deployed in QSR formats where a customer-facing ordering interface is combined with a cashier-facing management screen. It runs Google-certified Android with full GMS support, enabling complete AMAPI enrollment and the full Android Enterprise policy set. The device supports both single-screen and dual-screen configurations, the latter enabling simultaneous display of the ordering application on the customer-facing screen and order management functions on the operator-facing screen, managed as a single enrolled device with separate application assignments per display.

From a software architecture perspective, the dual-screen Sunmi T2 configuration requires careful MDM policy design. The primary display kiosk application must be set to KIOSK installType to engage lock task mode on the customer-facing screen, while the operator-facing screen runs a separate application that requires FORCE\_INSTALLED rather than KIOSK installType to allow the operator to navigate between management functions. This

split-installType policy configuration is one of the more technically nuanced aspects of Sunmi T2 MDM deployment and a common source of misconfiguration in the field.

### 5.4 Samsung QMR-B Commercial Display: Digital Menu Board and Drive-Through Controller

The Samsung QMR-B series commercial display is the dominant platform for digital menu boards, drive-through order confirmation displays, and customer-facing promotional signage in major QSR brands. Unlike the POS and kiosk devices discussed above, QMR-B displays run Samsung's Tizen operating system in their native configuration, but the QMR-B models designated for Android Enterprise management run a variant of the Samsung Smart Signage Platform (SSSP) built on Android.

Samsung Knox is deeply integrated into the QMR-B Android platform, providing the hardware-backed security and management extensions that differentiate Samsung commercial displays from consumer-grade alternatives. Knox Mobile Enrollment enables zero-touch provisioning of QMR-B displays at scale, and Knox E-FOTA provides firmware-level update management that allows IT administrators to target specific Samsung firmware builds rather than simply accepting the next available OTA update. This firmware-level control is particularly valuable in QSR deployments where digital menu board content management applications have specific version dependencies on the underlying Android platform.

Specification	Samsung QMR-B Commercial Display
Operating System	Android with Samsung SSSP platform
Knox Integration	Knox Platform for Enterprise, Knox Mobile Enrollment, Knox E-FOTA
OEMConfig	Samsung Knox Service Plugin (KSP) via OEMConfig
Update Management	Knox E-FOTA for firmware-level targeting
Zero-Touch	Knox Mobile Enrollment for large-scale provisioning
Security	Hardware-backed key storage, Samsung TrustZone
Display Options	32-inch to 85-inch diagonal, portrait or landscape
Primary QSR Use	Digital menu boards, drive-through displays, promotional signage

### 5.5 Zebra TC53: Rugged Handheld for Staff Operations

The Zebra TC53 is the current-generation successor to the widely deployed TC52, offering a Qualcomm Snapdragon processor, 6GB RAM, 128GB storage, Android 13 with LifeGuard update support, and Zebra's SE55 scanning engine capable of reading damaged, wet, or poorly printed

barcodes at distances up to 70cm. The TC53 is IP67-rated for dust and water resistance and MIL-STD-810H certified for drops, vibration, and temperature extremes, making it appropriate for kitchen and back-of-house environments in QSR contexts.

Zebra devices support Android Enterprise enrollment through standard AMAPI enrollment paths on GMS variants. However, Zebra's OEMConfig application, called Zebra OEMConfig, exposes a significantly richer management surface than the base Android Enterprise policy set through the MX (Mobility Extensions) framework. MX profiles cover USB configuration, scanning engine parameter tuning, display brightness and timeout management, Wi-Fi roaming behavior, and the StageNow provisioning integration that enables zero-touch-like enrollment for both GMS and non-GMS Zebra devices. Zebra's LifeGuard for Android update program commits to up to eight years of OS security patch support for enterprise device lines, which is the longest such commitment in the Android commercial device ecosystem and directly relevant to PCI DSS v4.0.1 patch currency requirements.

Specification	Zebra TC53
Processor	Qualcomm Snapdragon, octa-core
RAM and Storage	6GB RAM, 128GB internal flash storage
Operating System	Android 13, LifeGuard update program
Security Update Commitment	Up to 8 years from device launch
Scanning Engine	Zebra SE55, 1D and 2D barcodes, damaged label reading
Durability	IP67 dust and water, MIL-STD-810H drop and vibration certified
OEMConfig	Zebra OEMConfig with MX (Mobility Extensions) framework
Enrollment	AMAPI (GMS builds), StageNow (GMS and AOSP builds)
Primary QSR Use	Inventory management, order taking, back-of-house operations

## 6. Android Enterprise Software Architecture

### 6.1 The DevicePolicyManager API and Device Owner Mode

The foundation of Android enterprise device management is the DevicePolicyManager (DPM) API and the Device Owner (DO) privilege level it grants to enrolled MDM applications. A Device Owner application has system-level control over the enrolled device that exceeds what any standard application can obtain, including the ability to set persistent device policies, restrict or allow specific applications and hardware capabilities, control network configuration, and manage system update behavior. Achieving Device Owner status requires provisioning through one of the Android Enterprise enrollment paths: QR code provisioning, NFC bump provisioning, zero-touch enrollment, or the DPC identifier provisioning method.

The AMAPI framework delegates Device Owner privileges to the Android Device Policy application, Google's system-level DPC that is installed during enrollment and acts as the local policy enforcement agent on the device. When an MDM platform pushes a policy update through the AMAPI REST API, the policy is delivered to Android Device Policy through FCM, which then invokes the appropriate DevicePolicyManager methods to apply the policy to the device. This architecture means that MDM platforms built on AMAPI do not need to maintain their own DPC application but are constrained to the policy capabilities that AMAPI exposes through its policy resource schema.

### 6.2 The AMAPI Policy Resource and Dedicated Device Configuration

The AMAPI policy resource is the central configuration object that governs the behavior of enrolled devices. For QSR dedicated device deployments, the most critical policy fields are those that control application installation mode, system navigation, hardware capability access, and system update behavior. The Google AMAPI dedicated devices documentation, updated March 2026, defines the following key policy fields for dedicated device configurations:

AMAPI Policy Field	Type	QSR Application
installType: KIOSK	ApplicationPolicy	Launches app full-screen on boot, enables lock task mode, pins navigation
kioskCustomLauncherEnabled	Boolean	Allows multiple apps from a managed home screen instead of single-app lock
safeBootDisabled: true	Boolean	Prevents users from booting into safe mode to bypass kiosk lock
factoryResetDisabled: true	Boolean	Blocks factory reset from device settings, critical for POS terminals
screenCaptureDisabled: true	Boolean	Prevents screen capture on payment terminals (PCI DSS requirement)

AMAPI Policy Field	Type	QSR Application
cameraDisabled: true	Boolean	Disables camera except where explicitly required by application
systemUpdate: WINDOWED	SystemUpdate	Restricts OTA updates to configured maintenance window (e.g. 2am to 4am)
startMinutes / endMinutes	Integer	Defines the maintenance window for WINDOWED update type
keyguardDisabled: true	Boolean	Keeps device unlocked for customer-facing kiosk applications

The distinction between installType KIOSK and FORCE\_INSTALLED is the most technically consequential configuration choice in QSR dedicated device MDM deployment. KIOSK installType invokes the Android lock task mode API through DevicePolicyManager.setLockTaskPackages(), which pins the designated application to the foreground at the OS level. If the application crashes, the OS automatically relaunches it. If the user attempts to exit the application, the navigation is blocked at the system level. FORCE\_INSTALLED simply ensures the application is installed and present on the device but does not invoke lock task mode, meaning the device remains navigable to the Android home screen and application crashes are not automatically recovered.

### 6.3 Zero-Touch Enrollment Architecture

Zero-touch enrollment (ZTE) is the Android Enterprise enrollment mechanism that allows devices to be provisioned as fully managed or dedicated devices automatically when they connect to a network for the first time, without requiring any manual interaction with the device setup wizard. The mechanism works through a pre-provisioning record created in the Zero-Touch Enrollment portal, which is associated with the device's IMEI or serial number by the reseller or OEM at purchase time. When the device boots and connects to the network, it contacts the Zero-Touch provisioning server, retrieves the enrollment configuration, downloads and installs the designated DPC, and completes enrollment automatically.

For QSR fleet deployments, ZTE provides significant operational value because it eliminates the per-device provisioning labor that would otherwise be required for large-scale rollouts. A QSR brand deploying 500 kiosks across a franchise expansion can ship devices directly from the OEM or distributor to franchise locations, and each device will self-enroll and self-configure when powered on, including installing the designated applications, applying the kiosk policy, and connecting to the MDM console for monitoring. Samsung Knox Mobile Enrollment extends this concept to Samsung devices with additional configuration capabilities, including the ability to set the enrollment configuration before the Android Setup Wizard runs, which provides an even more seamless provisioning experience for Samsung-dominant fleets.



## 7. MDM Policy Implementation and Kiosk Mode

### 7.1 Lock Task Mode: Technical Implementation

Lock task mode is the Android system mechanism that restricts a device to a whitelist of approved applications and prevents the user from navigating outside those applications using standard Android navigation controls. It is invoked through the `DevicePolicyManager.setLockTaskPackages()` API, which accepts an array of package names that are allowed to run in lock task mode. Applications that call `Activity.startLockTask()` when included in this whitelist are pinned to the foreground with the navigation bar and status bar hidden or restricted.

When AMAPI sets an application's `installType` to `KIOSK`, Android Device Policy automatically calls `setLockTaskPackages()` to include that application's package name in the lock task whitelist and calls `startLockTask()` when the application launches. This creates an automatically recovering kiosk environment: if the application process terminates unexpectedly, the OS relaunches it and calls `startLockTask()` again, returning the device to the kiosk state without requiring IT intervention or physical device access.

The AMAPI dedicated device policy example from the Google developer documentation illustrates the minimum policy configuration for a production QSR kiosk deployment:

```
"safeBootDisabled": true,
"screenCaptureDisabled": true,
"factoryResetDisabled": true,
"cameraDisabled": true,
"systemUpdate": {
  "type": "WINDOWED",
  "startMinutes": 120,
  "endMinutes": 240
},
"applications": [{
  "packageName": "com.brand.ordering",
  "installType": "KIOSK"
}]
```

The `startMinutes` value of 120 and `endMinutes` value of 240 define a maintenance window from 2:00 AM to 4:00 AM local time, during which the system will automatically apply available OTA updates. This window corresponds to the lowest-traffic period for most QSR formats and minimizes the risk of update-related service disruption during operational hours.

### 7.2 Application Package Management in MDM Deployments



Android application distribution in enterprise MDM contexts occurs through Managed Google Play, the enterprise variant of the Google Play application distribution infrastructure. Managed Google Play allows IT administrators to approve applications from the public Play Store, upload private APKs for internal distribution, and push approved applications to enrolled devices through the AMAPI application policy. Applications distributed through Managed Google Play are signed with the same signing key as their public Play Store equivalents, ensuring code integrity verification through the Android package manager's signature check.

For QSR deployments, the application management architecture must address several constraints not present in general enterprise contexts. POS and payment applications are frequently distributed as private APKs through the MDM platform rather than through the public Play Store, because their payment credentials and configuration are embedded in the build and are not appropriate for public distribution. Kitchen display applications are often designed for a specific hardware platform and display configuration and must be distributed as platform-specific builds. The AMAPI application policy supports both Managed Google Play distribution and private APK hosting through the Google Play custom app publishing API, which is the standard mechanism for distributing payment and QSR-specific applications through Esper-managed deployments.

### **7.3 Remote Control and Device Diagnostics**

Remote control capability is the MDM feature with the highest direct operational impact in QSR deployments because it determines whether a crashed or misconfigured device can be diagnosed and remediated without dispatching a technician. The technical implementation of remote control on Android differs significantly between MDM platforms and has meaningful implications for QSR deployment architecture.

Esper's remote control implementation uses a device-side agent that streams display output over a WebRTC connection and accepts input injection through the AccessibilityService API, which does not require the device to be rooted or to have a custom Android build. This approach works on any Android device running Android 8 or higher with the Esper agent installed. Microsoft Intune's remote control depends on TeamViewer Quick Support integration, which requires TeamViewer to be installed and activated on the managed device, adding a dependency that is not present in purpose-built implementations. SOTI MobiControl's remote control uses a similar agent-based approach to Esper but with a proprietary transport protocol rather than WebRTC.

The technical distinction matters in QSR deployments because the remote control session must remain available even when the primary kiosk application is locked to the foreground in KIOSK installType mode. Implementations that rely on the device's standard Android UI may be blocked by the kiosk lock. Esper's implementation bypasses this constraint because the remote

control agent operates below the application layer, at the level of the Android display service and input subsystem, making it accessible regardless of the application state.

## 8. OEM-Specific Management Extensions

### 8.1 Samsung Knox OEMConfig and the Knox Service Plugin

Samsung Knox OEMConfig, implemented through the Knox Service Plugin (KSP), is the most comprehensive OEM management extension in the Android enterprise ecosystem. KSP exposes over 1,000 Samsung-specific policy parameters through the standard OEMConfig managed configuration framework, making them accessible from any AMAPI-compatible MDM platform without requiring Samsung Knox Manage or any other Samsung-proprietary management console.

The KSP policy parameters of greatest relevance to QSR deployments fall into four categories. Display and power management parameters include screen timeout configuration beyond the standard Android DPM limits, display brightness scheduling by time of day, and power button behavior customization to prevent accidental shutdown of customer-facing kiosks. Network management parameters include Wi-Fi roaming aggressiveness tuning, which is relevant in large QSR locations where devices may move between multiple access points, and cellular data policy enforcement for devices with SIM cards. Hardware peripheral control includes USB port configuration, allowing IT administrators to permit charging but block data transfer on customer-facing kiosk USB ports, which is both a PCI DSS v4.0.1 requirement and a physical security control. The Knox E-FOTA firmware update system provides the most operationally important capability: the ability to target a specific Samsung firmware build for deployment rather than accepting any available update.

### 8.2 Knox E-FOTA: Firmware-Level Update Management

Knox E-FOTA (Enterprise Firmware Over The Air) is Samsung's enterprise update management system that provides control over firmware updates at a level of specificity that the base AMAPI `systemUpdate` policy does not achieve. While AMAPI's `systemUpdate WINDOWED` policy restricts when updates are applied, it does not allow the administrator to specify which update version should be applied. A device on Samsung firmware version R354XXU5CWJA might receive either of two available updates during its maintenance window, with no MDM-level control over which version is installed.

Knox E-FOTA resolves this by allowing IT administrators to create update policies that target a specific firmware build string. The policy can be configured to hold devices at the current firmware version indefinitely, to allow only approved firmware versions, or to enforce a specific firmware version across the fleet. For QSR operators managing restaurant management system integrations that have documented dependencies on specific Samsung firmware builds, Knox E-FOTA is not a convenience feature but a production stability requirement. This capability is

accessible through Esper's Knox E-FOTA integration, which exposes the E-FOTA policy configuration through the Esper console for Samsung-dominant fleets.

### 8.3 Sunmi OEMConfig

Sunmi's OEMConfig application exposes Sunmi-specific device management parameters through the standard Android managed configuration framework. The parameters relevant to QSR deployments include the Sunmi system launcher configuration, which controls what applications appear on the Sunmi home screen before KIOSK mode application install completes; the SunmiOS update channel, which allows IT administrators to pin devices to a specific Sunmi firmware channel; hardware peripheral settings including printer paper width, print density, and auto-cut behavior for the integrated thermal printer; and the Sunmi Pay integration configuration for devices with embedded payment modules.

A technically important aspect of Sunmi OEMConfig in QSR deployments is the interaction between the Sunmi system launcher and AMAPI's KIOSK installType. When a Sunmi device first boots after enrollment, the Sunmi launcher is the default home application. The AMAPI policy's KIOSK installType instructs Android Device Policy to set the kiosk application as the default home application through the DevicePolicyManager.addPersistentPreferredActivity() API, replacing the Sunmi launcher. If the OEMConfig configuration has not been applied before this API call, the Sunmi launcher may conflict with the KIOSK application launch, resulting in a race condition that leaves some devices in a partially configured state. The correct deployment sequence is to apply OEMConfig configuration first, confirm its application, and then push the KIOSK application policy.

## 9. Real-World Deployment Observations

### 9.1 The FORCE\_INSTALLED vs. KIOSK Misconfiguration Pattern

The most frequently observed technical misconfiguration in QSR MDM deployments reviewed during the research period was the use of FORCE\_INSTALLED installType for the primary kiosk application rather than KIOSK. This misconfiguration was present in approximately one-third of the deployments reviewed, consistently in operators who had migrated from a general-purpose EMM platform that had configured the ordering application as a mandatory enterprise application using the enterprise app management framework designed for knowledge-worker device management.

The operational consequence of this misconfiguration was a pattern of kiosk devices that intermittently displayed the Android home screen, a situation reported by franchise operators as 'kiosks going to the desktop' or 'customers seeing the Android menu.' The root cause was always the same: the ordering application had crashed or been backgrounded, and because lock task mode had not been activated through KIOSK installType, Android's standard application lifecycle management allowed other applications or the home screen to take focus. Remediation required updating the AMAPI policy to change installType to KIOSK and pushing a policy refresh to enrolled devices, which resolved the issue without physical device access.

### 9.2 OTA Update Failures and API Level Constraints

The second most common technical issue observed was OTA update failures caused by API level incompatibilities between updated application packages and the Android version running on target devices. In one representative case reviewed during the research period, a QSR brand had updated their ordering application to target Android API level 33 (Android 13), which introduced a behavior change in the application's use of the Notification.Builder API that was only compatible with devices running Android 12 or higher. Devices in the fleet still running Android 11 (API level 30) installed the updated APK but crashed on launch when the Notification.Builder call attempted to use a constant only defined in API 31 and above.

The MDM-level resolution required configuring a minimum API level constraint in the Managed Google Play application policy, which prevented the updated APK from being pushed to devices below API level 31, combined with a separate OTA update campaign to bring API level 30 devices to Android 12 before the application update was applied. This sequencing required the MDM platform to support both API level-conditional application deployment and staged OTA update orchestration, capabilities that not all MDM platforms implement with equal specificity.

### 9.3 The Zero-Touch Enrollment Timing Problem

Zero-touch enrollment functions correctly when devices are connected to the internet immediately after factory reset or first boot. In QSR franchise deployments, devices are sometimes powered on in a warehouse or staging area that is connected to a network that does not have access to the Google zero-touch provisioning server at [mtalk.google.com](https://mtalk.google.com), which requires outbound TCP connectivity on port 5228. In these cases, the device completes the Android Setup Wizard without enrolling, leaving it in an unmanaged state that requires re-flashing or a manual QR code enrollment to correct.

The MDM-level mitigation for this is to pre-configure devices with a static Wi-Fi network credential that is trusted to have correct internet access, which can be embedded in the zero-touch enrollment configuration record. Esper's zero-touch configuration supports embedding Wi-Fi credentials in the enrollment record so that devices automatically connect to the pre-configured network before attempting provisioning server contact, eliminating the staging network dependency.

## 10. OS Fragmentation and Security Patch Analysis

### 10.1 Android API Level Distribution in QSR Deployments

The distribution of Android API levels across active QSR deployments, observed across the Esper-managed fleet during the research period, reveals a concentration of devices at API levels that limit available MDM policy capabilities and create PCI DSS v4.0.1 exposure through expired security patch support.

Android Version	API Level	Est. QSR Fleet Share	Key MDM Capability Limitations
Android 8.x (Oreo)	26 to 27	Approx. 6%	No AMAPI dedicated device support. Must use legacy DAA enrollment
Android 9 (Pie)	28	Approx. 12%	Basic AMAPI support. No kioskCustomLauncherEnabled policy field
Android 10 (Q)	29	Approx. 20%	Most AMAPI dedicated device fields supported. No WINDOWED update in all OEMs
Android 11 (R)	30	Approx. 22%	Full dedicated device AMAPI. Enhanced lock task APIs. Knox E-FOTA v2 required
Android 12 and 12L (S)	31 to 32	Approx. 19%	Advanced Protection Mode preview. Improved kiosk launcher stability
Android 13 (T)	33	Approx. 12%	Full AMAPI feature set. Photo and media permission granularity improved
Android 14 (U)	34	Approx. 7%	Identity Check preview. Health Connect integration for device diagnostics
Android 15 and 16	35 and above	Approx. 2%	Advanced Protection Mode GA. Identity Check GA. AMAPI MCP preview

The concentration of approximately 38 percent of QSR Android deployments on Android 9 and 10, both of which have reached end of Google Android security update support and are approaching or past end of OEM security patch support for most commercial hardware lines, represents a systematic PCI DSS v4.0.1 compliance gap. PCI DSS v4.0.1 Requirement 6.3.3 requires that all system components are protected from known vulnerabilities by installing applicable security patches and updates, with critical patches installed within one month of release. Devices on Android 9 or 10 that are no longer receiving security patches are in structural non-compliance with this requirement regardless of other controls.

### 10.2 PCI DSS v4.0.1 and the Patch Currency Problem

The PCI DSS v4.0.1 standard, which became the sole active version for compliance assessments after December 31, 2024, introduced expanded requirements for mobile and IoT devices in

payment processing environments. Requirement 12.3.4 now mandates that hardware and software technologies are reviewed at least once every 12 months to confirm whether they continue to receive security fixes from vendors, and that appropriate action is taken when technologies no longer receive fixes. For QSR operators with Android 9 or 10 devices in payment processing scope, this requirement creates a documented obligation to remediate or isolate those devices.

The MDM platform plays a direct role in PCI DSS compliance management through two capabilities. First, the platform's device inventory and OS version reporting must provide accurate, current patch level data that can be included in compliance evidence. AMAPI provides a securityPosture field in the device resource that includes the common criteria mode and device posture assessment, and the devices.get API response includes the last reported security patch level, which compliance teams can extract through the MDM platform's reporting APIs. Second, the MDM platform's OTA update management must support the controlled upgrade campaigns required to bring non-compliant devices to a patched OS version without disrupting production operations.



## 11. Emerging Technical Developments

### 11.1 Android 16 Enterprise API Changes

Android 16, with general availability expected in Q3 2026, introduces several changes to the Android Enterprise API surface that are directly relevant to QSR MDM implementations. The Advanced Protection Mode API, accessible through `DevicePolicyManager.setAdvancedProtectionEnabled()` on enrolled fully managed devices, enforces a set of hardware-level security restrictions that go beyond standard Android Enterprise policy: USB data transfer is blocked at the hardware controller level, not just through policy; sideloading of APKs from unknown sources is blocked even for Device Owner applications; and attestation-based network access control is enforced for connections to configured enterprise services.

The Identity Check feature, now a stable API in Android 16, requires biometric authentication before allowing modification of device management settings, unenrollment, or factory reset. At the implementation level, Identity Check intercepts the `DevicePolicyManager.removeActiveAdmin()` and `wipeData()` method calls and gates them behind a biometric authentication challenge. For QSR kiosk deployments, this provides a hardware-enforced defense against physical tampering with payment terminals in retail environments where unauthorized factory reset is a documented fraud vector.

The AMAPI MCP (Model Context Protocol) preview, documented in the April 2026 AMAPI release notes, provides a foundation for AI assistant integration with device management data. The MCP server exposes AMAPI device resource data through a structured query interface, enabling queries such as 'which devices in my QSR fleet are running an outdated security patch level' without requiring custom reporting queries against the AMAPI REST API. For QSR IT teams managing large fleets with limited engineering resources, this capability has practical value for compliance monitoring and anomaly detection.

### 11.2 Knox E-FOTA and Predictive Device Health

Samsung Knox Asset Intelligence, available within the Knox Suite, collects device telemetry including battery health cycles, CPU temperature trends, storage fragmentation levels, and application crash frequency, and makes this data available through the Knox Asset Intelligence API for integration with MDM platform analytics. The technical architecture uses a device-side agent that samples hardware metrics at configurable intervals and uploads them to the Knox cloud backend, which aggregates the data across the managed fleet and exposes it through a REST API that Esper and other Knox-integrated MDM platforms can consume.

The predictive value of this telemetry is most clearly demonstrated in battery degradation monitoring. Android devices report battery health through the `BatteryManager` API, which

exposes current capacity versus design capacity as a percentage. Devices below approximately 80 percent battery health begin to exhibit unpredictable shutdown behavior under load, which in a QSR kiosk context manifests as unexpected device resets during peak service periods. The Knox Asset Intelligence platform can alert when devices cross configurable battery health thresholds, allowing IT administrators to schedule preventive battery replacement before the device fails in production.

## 12. Technical Framework for MDM Selection

### 12.1 Technical Evaluation Criteria

Based on the technical analysis in the preceding sections, the following framework defines the criteria that should drive MDM platform selection for QSR dedicated-device deployments. Each criterion is grounded in a specific technical requirement identified in this paper.

Technical Criterion	Weight	Specific Requirement
KIOSK installType implementation	25%	Must invoke OS-level lock task mode, not application-layer pinning
OTA orchestration granularity	20%	Must support staged rollout with API level targeting and rollback policy
Remote control below application layer	20%	Must function while KIOSK installType lock task mode is active
OEMConfig support breadth	15%	Must support Samsung KSP, Sunmi OEMConfig, and Zebra OEMConfig
Zero-touch enrollment with Wi-Fi pre-seeding	10%	Must support embedding Wi-Fi credentials in ZTE enrollment config
PCI DSS patch reporting	10%	Must expose security patch level via reporting API for compliance evidence

### 12.2 Platform Technical Comparison

Platform	KIOSK installType	OTA Granularity	Remote Control in Kiosk	OEMConfig Breadth	ZTE Wi-Fi	Patch Reporting
Esper	Native AMAPI KIOSK	Stage and rollback supported	Below app layer via agent	Samsung, Sunmi, Zebra	Supported	API exposed
Intune	AMAPI KIOSK via Intune	Basic windowed, no staging	TeamViewer dependency	Samsung KSP only	Supported	Compliance dashboard
SOTI MobiControl	AMAPI KIOSK	Basic windowed	Agent-based, in-kiosk capable	Samsung, Zebra	Supported	Report builder
Workspace ONE	AMAPI KIOSK	Basic windowed	TeamViewer dependency	Samsung, Zebra	Supported	Intelligence module
Jamf	Limited Android support	Basic only	Limited	Limited	Partial	Basic only

## 13. Conclusions

Android device management in the QSR industry is a computer science problem that requires technical depth to solve correctly. The hardware diversity of the QSR Android ecosystem, spanning PAX PayDroid payment terminals, Sunmi GMS-certified kiosks, Samsung Knox commercial displays, and Zebra OEMConfig-extended rugged handhelds, creates a management surface that demands both breadth of platform API support and specific technical knowledge of how each OEM implements its management extension layer.

The AMAPI dedicated device policy model, and specifically the KIOSK installType field that invokes OS-level lock task mode, is the foundational technical mechanism that separates stable QSR kiosk deployments from fragile ones. The prevalence of FORCE\_INSTALLED misconfiguration observed during the research period is a direct consequence of general-purpose EMM platforms and the practitioners who configure them approaching kiosk deployment as an application management problem rather than an OS configuration problem.

OS fragmentation at API levels below 30 creates compounding technical debt: devices on Android 9 and 10 are simultaneously out of security patch support, unable to use key Android Enterprise APIs introduced in Android 11, and structurally non-compliant with PCI DSS v4.0.1 Requirement 6.3.3. The MDM platform's OTA update orchestration capability is therefore not a convenience feature but a compliance-critical system component that must be technically evaluated, not just checked off a feature list.

The emerging technical developments in Android 16, including the Advanced Protection Mode GA release, the Identity Check API, and the AMAPI MCP preview, represent a meaningful improvement in the management and security posture available to QSR operators. Operators making MDM platform decisions in 2026 should include each platform's timeline for adopting these Android 16 APIs as a forward-looking technical criterion, because the security and operational benefits of these features are directly relevant to the QSR deployment context.

---

## 14. Acknowledgments

The QSR IT architects, franchise technology officers, and MDM platform engineers whose technical conversations during the November 2025 to May 2026 research period contributed directly to the findings in this paper. The specificity of the technical analysis in Sections 7, 8, and 9 reflects their willingness to discuss deployment failures and misconfiguration patterns with candor.

The Google Android Enterprise developer documentation team and the Samsung Knox documentation team, whose technical references are maintained with a level of precision and

currency that makes practitioner research of this kind possible. The AMAPI dedicated device policy documentation and the Samsung Knox Service Plugin technical guide were the primary technical references for Sections 6 through 8.

The University of Washington MCCN program for providing the academic framework within which practitioner technical experience is organized into structured research output.

## 15. References

- Google. (2026, March 5). Dedicated devices overview. Android Enterprise developer documentation. <https://developer.android.com/work/dpc/dedicated-devices>
- Google. (2025, March 5). Example policies: dedicated devices. Android Management API. <https://developers.google.com/android/management/policies/dedicated-devices>
- Google. (2026, April 7). Android Management API release notes. Google for Developers. <https://developers.google.com/android/management/release-notes>
- Google. (2026). REST resource: enterprises.policies. Android Management API reference. <https://developers.google.com/android/management/reference/rest/v1/enterprises.policies>
- Google. (2026, March 5). Device management overview. Android Open Source Project. <https://source.android.com/docs/devices/admin>
- Samsung. (2026). Knox Platform for Enterprise: Technical documentation and KSP OEMConfig reference. Samsung Electronics Co., Ltd. <https://docs.samsungknox.com/admin/knox-platform-for-enterprise/>
- Samsung. (2026). Knox Asset Intelligence: Device telemetry and analytics documentation. Samsung Knox Documentation. <https://docs.samsungknox.com/admin/fundamentals/>
- Samsung. (2026). Knox Mobile Enrollment: Zero-touch provisioning guide. Samsung Knox Documentation. <https://docs.samsungknox.com>
- PAX Technology. (2026). A920Pro Duo and A920 series: Product specifications and PayDroid platform documentation. PAX Technology, Inc. <https://www.pax.us/product/a920pro-duo/>
- Sunmi. (2026). K2 self-service kiosk: Product specifications and OEMConfig guide. Sunmi Technology Co., Ltd. <https://www.sunmi.com/en/k2/>
- Sunmi. (2026). D3 Pro: Smart desktop terminal specifications. Sunmi Technology Co., Ltd. <https://www.sunmi.com/en/d3-pro/>
- Zebra Technologies. (2025). TC52 and TC57 series touch computer specification sheet. Zebra Technologies Corporation. <https://www.zebra.com/us/en/products/spec-sheets/mobile-computers/handheld/tc52-tc57.html>
- Zebra Technologies. (2025). TC5 series mobile computers: LifeGuard for Android update program documentation. Zebra Technologies Corporation. <https://www.zebra.com/us/en/products/mobile-computers/handheld/tc5x-series.html>

- Bayton, J. (2026, March 22). Android Enterprise provisioning methods.  
<https://bayton.org/android/android-enterprise-provisioning-methods/>
- Bayton, J. (2026, February 19). Introducing AMAPI Commander: converse with your Android estate.  
<https://bayton.org/blog/2026/02/introducing-amapi-commander/>
- PCI Security Standards Council. (2024). PCI DSS v4.0.1: Payment Card Industry Data Security Standard requirements and testing procedures, including Requirement 6.3.3 and 12.3.4. PCI SSC.  
<https://www.pcisecuritystandards.org>
- Gartner. (2026). Magic Quadrant for Endpoint Management Tools. Gartner, Inc.  
<https://www.gartner.com/en/documents/7298830>
- Canopy. (2025). Fast-food friction: The 2025 Restaurant Tech Report. Coverage and findings reported in Nation's Restaurant News, August 18, 2025.  
<https://www.nrn.com/quick-service/report-80-of-consumers-believe-good-technology-is-important-when-choosing-a-qsr>
- Rosper Tech. (2026, April 9). How to set up SUNMI kiosk mode: Default startup app for self-ordering. <https://blog.rospertech.com/sunmi-kiosk-mode-default-startup-app/>