Artifact Evaluated NDSS Available

Beyond the App Store: Uncovering the Underground iOS Signing Supply Chain via Operational Data

Yicheng Hu
Zhejiang University
12521215@zju.edu.cn

Haitao Xu✉
Zhejiang University
haitaoxu@zju.edu.cn

Shuai Hao
Old Dominion University
shao@odu.edu

Chuan Yue
Colorado School of Mines
chuanyue@mines.edu

Wenrui Ma
Zhejiang Gongshang University
mawenrui@zjgsu.edu.cn

Zhenyuan Li
Zhejiang University
lizhenyuan@zju.edu.cn

Zhao Li
Hangzhou Yugu Technology
lzjoey@gmail.com

***Abstract*—Apple's iOS ecosystem relies on a centralized trust model where the App Store enforces strict app review. To support legitimate testing and enterprise deployment, Apple provides two alternative distribution mechanisms: ad hoc distribution and in-house distribution. Despite their benign intent, these mechanisms have been persistently abused, giving rise to a layered underground supply chain—the iOS Abusive Signing-Provisioning-Distribution (*iASPD*) ecosystem—that enables large-scale distribution of policy-violating and malicious apps while bypassing App Store review and revocation enforcement. Prior work has examined individual abuse vectors (*e.g.*, ad hoc sideloading, enterprise certificate misuse) but relied primarily on external observations, leaving the ecosystem's internal structure and adaptive strategies largely opaque. In this paper, we present an unprecedented inside view based on seized internal operational databases from multiple actors within this ecosystem, including service frontends, intermediaries, and upstream suppliers. Our dataset spans 258 backend databases (177 second-level domains) and is complemented by records from 60 e-commerce merchants and online conversations with 10 service providers. Using this unique data, we make three key findings. First, we characterize the apps and actors involved: ad hoc distribution primarily serves end users seeking unauthorized apps, while in-house distribution is favored by developers distributing policy-violating software at scale; among 22,259 unique apps, 18,555 (83.4%) are abusive, and 1,117 (49.9%) of 2,238 signed apps are malicious. Second, we uncover the multi-tier supply chain, showing that signing capacity is brokered through intermediaries, with higher-tier intermediaries capturing up to 72.6% of the revenue. Third, we reveal how upstream suppliers accumulate accounts (via leaked, fake identities, or account trading) and how the ecosystem survives certificate revocation through replacement and rerouting. Our findings demonstrate that unofficial iOS distribution is an organized, resilient service ecosystem. We provide a structural understanding of its layered market, monetization paths, and adaptive strategies, offering actionable insights for platform defenders and law enforcement. We have responsibly reported these findings to Apple.**

✉ Corresponding author: Haitao Xu (haitaoxu@zju.edu.cn).

Network and Distributed System Security (NDSS) Symposium 2027
22–26 March 2027, Seoul, Republic of Korea
ISBN 978-1-970672-09-1
https://dx.doi.org/10.14722/ndss.2027.230373
www.ndss-symposium.org

I. Introduction

The App Store serves as the official and most trusted channel for iOS app distribution, where Apple enforces rigorous app review, code signing, and ongoing security monitoring. To support legitimate development workflows, such as internal testing before public release or enterprise-specific app deployment, Apple provides two alternative distribution mechanisms. *Ad hoc distribution* (*i.e.*, ad hoc signing) [1] allows developers to install an app on up to 100 registered devices (identified by their UDIDs) for small-scale testing. *In-house distribution* (*i.e.*, enterprise signing) [2] permits an organization to deploy apps to an unlimited number of its own devices without App Store review. Both mechanisms rely on the same underlying signing and provisioning infrastructure: a valid *developer account*, a *signing certificate*, and a *provisioning profile* that binds the certificate, the app ID, and (in case of ad hoc) a device list.

Despite their legitimate purposes, these mechanisms have been persistently abused. Enterprise signing, which imposes no device limit, has become favored by developers seeking large-scale distribution of policy-violating or malicious software. Conversely, ad hoc signing, which ties an unlimited number of apps to a limited set of registered devices, has been repurposed to serve end users seeking to install multiple unauthorized apps on personal devices. This abuse has given rise to a layered underground supply chain that exploits Apple's entire signing-provisioning-distribution pipeline, from certificate acquisition and profile generation, to on-device installation, signature verification, and OCSP (Online Certificate Status Protocol)-based revocation checks. We term it the *iOS Abusive Signing-Provisioning-Distribution (iASPD)* ecosystem.

Within the *iASPD* ecosystem, *upstream suppliers* hoard developer accounts and signing materials; *intermediaries* broker signing capacity (*i.e.*, pools of active, unrevoked certificates) for profit; *service frontends* provide app-signing and hosting interfaces; and *promoters* attract developers and end users via social media and e-commerce platforms. The structure is highly resilient to enforcement: when Apple revokes a compromised certificate, an action that would normally terminate all apps signed with it, end users can simply request replace-

ment services at frontends, intermediaries reroute that signing request through reseller chains, upstream suppliers simply allocate a fresh certificate from their pool, and end users ultimately receive the same app signed with a new unrevoked certificate, effectively circumventing revocation and preserving the underground market. Apple's 2024 fraud prevention report [3] indirectly reflects the severity of this problem: the company prevented nearly 4.6 million attempts to install or launch apps from outside the App Store or other approved marketplaces, blocked over 10,000 illegitimate apps on pirate storefronts (including malware, pornography, gambling apps, and pirated copies of legitimate apps), terminated more than 146,000 developer accounts over fraud concerns, and rejected an additional 139,000 developer enrollments from bad actors.

Prior work has examined various facets of unauthorized iOS app distribution. Zheng *et al.* [4] identified the abuse of enterprise and developer certificates and described the apps signed with these certificates. Guo *et al.* [5] extended the analysis to off-store channels, focusing on how Telegram is used to disseminate illicit apps. The closest prior study, Liu *et al.* [6], dissected the ad hoc sideloading gray market and showed how signing capacity is converted into a sellable resource; however, their analysis focuses on ad hoc signing only and draws evidence from user-facing sites, signing tools, and app categorization, without directly observing the internal workflows that coordinate parties within the ecosystem. Nevertheless, these studies rely primarily on external observations and therefore leave the iASPD ecosystem's topology, actor roles, and cross-tier interactions uncharacterized.

In contrast, our work draws on seized internal operational databases from multiple actors within the iASPD ecosystem, including promoters, service frontends, intermediaries, and upstream suppliers. This operator-side view directly reveals cross-layer interactions, the allocation and monetization of signing capacity, and coordinated recovery after revocation: workflows that prior work could only infer. We therefore advance beyond prior work by explaining the incentives behind each role, the logic of the ecosystem's layered structure, and the rapid resource reallocation that sustains service continuity.

To structure our analysis, we focus on a resource that our observations reveal as the ecosystem's central chokepoint: signing capacity. Its flow from upstream suppliers to service frontends, and its reallocation after revocation, determine both market resilience and effective intervention points. We therefore propose three research questions that map the ecosystem from the bottom up:

- **RQ1:** What drives the formation of the underground market, and what types of apps are distributed?
- **RQ2:** What makes signing capacity the market's central scarce resource, and how does it flow among different roles?
- **RQ3:** Where does signing capacity originate, and how does it enable the ecosystem to survive Apple's enforcement?

To answer these questions, we analyze 258 internal backend databases seized by Chinese law enforcement from underground iOS app distribution operations, spanning 177 second-level domains and covering Dec. 2021 to Mar. 2026. The databases capture the full market stack: 24 upstream suppliers, 103 intermediaries, and 131 service frontends. We supplement these with 60 e-commerce merchants and 10 online conversations with signing/hosting providers.

We conducted a systematic analysis based on these datasets. For RQ1, two demand groups drive the market: app developers seeking to distribute policy-violating or malicious software, and end users wanting unauthorized apps on personal devices. The distributed apps are dominated by cracked games, premium-unlocked apps, modified mainstream variants. For RQ2, signing capacity is the ecosystem's central scarce resource because each off-store installation requires valid, unrevoked signing materials whose availability is constrained by Apple's account and provisioning rules. It is thus repackaged and sold per device slot for ad hoc distribution and per app for enterprise distribution, then transferred through multi-tier intermediary reseller chains. For RQ3, signing capacity originates from upstream suppliers who pool developer accounts acquired through bulk registration using fake or leaked identities or account trading. When Apple revokes a certificate, replacement demand propagates upstream, a new certificate is allocated, fresh signing materials are returned, and frontends re-sign the app, turning revocation into an absorbable operational cost rather than a disruptive event. Overall, the iASPD ecosystem turns Apple's provisioning infrastructure into a resilient underground service, harming Apple's economic interests, exposing users to security and privacy risks, and enabling widespread malware distribution and copyright infringement. We responsibly disclosed our findings to Apple, which acknowledged the report and expressed interest in further discussion.

In summary, our contributions are as follows:

- **Operator-side ecosystem mapping.** Drawing on 258 seized internal operational databases, we reveal that unauthorized iOS distribution is a layered service market: developers and end users generate demand, promoters attract customers, service frontends provide signing and hosting, intermediaries resell signing capacity, and upstream suppliers maintain account pools. This inside view uncovers how demand generation, service delivery, resale, and account maintenance are operationally distinct across these roles, yet coordinated to sustain the overall ecosystem.
- **Characterization of abusive apps and malware.** We find that the ecosystem primarily distributes cracked games, premium-unlocked apps, and modified mainstream variants. Moreover, among signed IPAs, nearly half are flagged as malicious, with malware concentrated in gambling, unlicensed finance, and modified apps.
- **Resilience through resource replacement.** We show that Apple's distribution constraints shape underground products: ad-hoc signing is monetized as device-slot capacity, while enterprise signing as per-app distribution. Intermediaries hide upstream complexity via multi-tier routing, and upstream suppliers absorb revocation by allocating fresh certificates. Consequently, certificate revocation becomes a recoverable supply-chain event, not a decisive disruption.

TABLE I: Characteristics and known abuses of mainstream iOS app distribution and installation methods.

Method	Description	App Review?	Required Program	Constraints	App Validity	Known Abuses
App Store	Official distribution channel	Yes	Apple Developer Program ($99/year)	Strict app review	Permanent	Camouflaged (chameleon) apps [7]–[9]
TestFlight	Official beta testing	Yes		Strict beta-app review	90 days	Fraudulent beta invitations [10], [11]
Ad hoc	Ad-hoc distribution for testing	No		Manual UDID registration; 100 devices/year	1 year	Signing certificate resale [6], [12]
In-house	Enterprise in-house distribution	No	Apple Developer Enterprise Program ($299/year)	Internal use only; active monitoring and revocation	1 year (renewable)	Widespread enterprise certificate abuse [4], [13], [14]
WebClip	Web clip bookmark to webpage	No	None	Not a native app; limited functionality	Permanent	Phishing or scam websites [5], [15]

II. Background

In this section, we first summarize the major app distribution methods on iOS, and then describe the normal process of two distribution mechanisms, ad hoc distribution and in-house distribution, provided by Apple beyond the App Store.

A. iOS App Distribution Methods

Beyond the official App Store, Apple supports several alternative mechanisms for app installation on iOS devices. Table I summarizes five representative methods, including their intended purpose, review requirements, associated developer programs, operational constraints, app validity periods, and known abuse patterns.

App Store and TestFlight. These are Apple's official channels for public distribution and beta testing [16], [17]. Both require app review (full review for App Store; a dedicated beta review for TestFlight) and are subject to ongoing abuse detection after release [18]. Nevertheless, abusive cases have been documented, such as Camouflaged (chameleon) apps that evade review by dynamically altering behavior [7]–[9], and fraudulent beta invitations that trick users into installing malicious apps disguised as TestFlight pre-releases [10], [11].

Ad Hoc Distribution. This channel is intended for small-scale testing before public release [1]. It requires no App Store review but is limited to 100 registered devices per developer account per year [19]. It requires enrollment in the standard Apple Developer Program ($99/year), and enrollment demands identity verification (for individuals) or legal entity validation (*e.g.*, a D-U-N-S number for organizations) [20]. Apple may impose operational controls, such as delayed processing for new accounts or devices previously linked to a revoked account [21]. Users may also need to enable Developer Mode on newer iOS versions [22]. The primary known abuse of ad hoc distribution is the resale of signing certificates to install unauthorized apps on registered devices [6], [12].

In-House Distribution (Enterprise Signing). This channel allows enterprises to distribute proprietary apps internally to their employees without App Store review [2]. It requires the Apple Developer Enterprise Program ($299/year), and its enrollment is stricter: applicants must be verified legal enterprises with sufficient scale [20], [23]. Unlike ad hoc, in-house distribution imposes no device registration limit, enabling broad deployment. However, Apple actively monitors usage patterns and can quickly revoke certificates when distribution exceeds internal use [24]. The main documented abuse is widespread misuse of enterprise certificates to sign and distribute arbitrary apps, including malware and policy-violating software, to non-employee devices [4], [13], [14].

WebClip. A WebClip is not a native app distribution mechanism, but a home-screen shortcut to an external webpage [25]. It requires no developer program, no review, and offers no native API access. Because it is easy to generate and deploy, WebClips have been widely abused to direct users to phishing sites, scam pages, or other untrusted web content [5], [15].

Summary. Among these methods, ad hoc and in-house distribution are particularly significant for unauthorized sideloading, as they bypass ordinary App Store review while preserving a native-app installation experience. They therefore have become attractive both to developers seeking to distribute unauthorized apps and to end users seeking convenient access to such apps.

Between the two distributions, ad hoc is constrained by manual device registration and a limit of 100 devices per developer account per year, making it suitable for small-scale testing. In-house distribution imposes no device limit, enabling broad deployment to employees, but requires stricter enrollment (verified enterprises at scale) and is subject to active monitoring and rapid certificate revocation upon misuse.

B. Ad Hoc and In-House Distribution Workflow

Fig. 1 illustrates the end-to-end workflow for distributing iOS apps via ad hoc and in-house (enterprise) channels, from development to execution on an end user's device. The developer first builds an iOS app and packages it as an unsigned IPA (iOS Package Archive) (❶). Deployment outside the App Store still requires Apple-recognized code signing and mobile provisioning. The developer accordingly enrolls in the appropriate Apple developer program (❷): the standard Developer Program for ad hoc distribution, or the Enterprise Program for in-house distribution. Next, the developer registers an App ID, generates signing keys locally, and obtains the corresponding signing certificates from Apple (❸). For

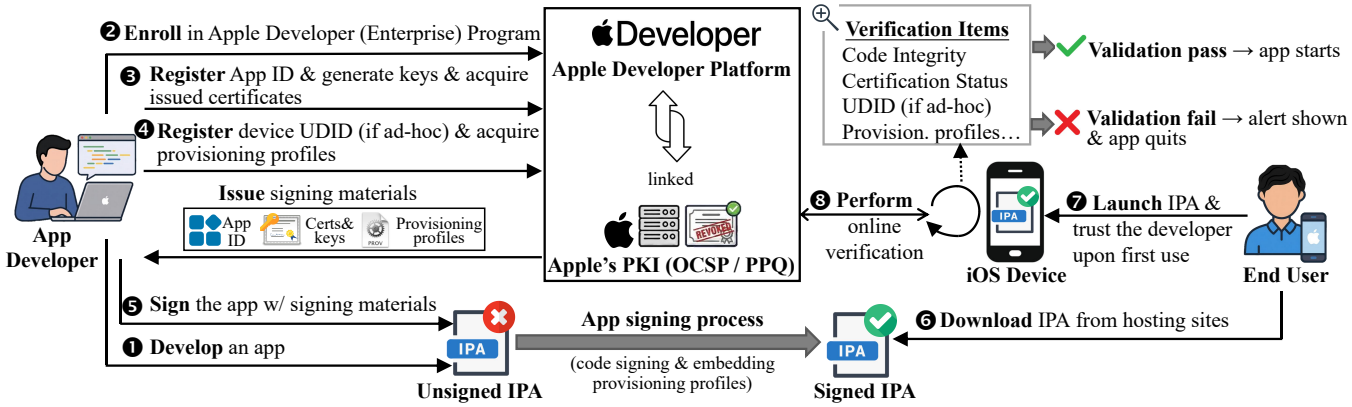


Fig. 1: Overview of the ad-hoc and in-house distribution process.

ad hoc distribution, the developer also registers the Unique Device Identifiers (UDIDs) of target test devices and acquires provisioning profiles that bind the certificate, App ID, and device list (❹). Together, the App ID, signing certificates, and provisioning profiles constitute the signing materials. Using these materials, the developer code-signs the app and embeds the provisioning profile, producing a signed IPA (❺).

The developer then uploaded the signed IPA to third-party hosting sites, from which end users can download it (❻). After downloading, the user launches the IPA on their iOS device and, on first use, explicitly trusts the developer certificate (❼). Before the app can start, the device performs online verification with Apple's Public Key Infrastructure (PKI) services (the Online Certificate Status Protocol (OCSP) for certificate status, Provisioning Profile Query (PPQ) for provisioning validation) (❽). The verification encompasses code integrity, certificate status, UDID matching (for ad hoc), and provisioning profile validity [26]. If all checks pass, the app launches successfully; otherwise, the system alerts and terminates the app. This flow highlights how Apple's online verification enforces distribution constraints even for apps installed outside the App Store.

III. Ecosystem Overview

The abuse of Apple's ad hoc and in-house distribution channels is driven by two distinct sources of demand: developers of policy-violating or malicious software seeking large-scale distribution, and end users seeking to install multiple unauthorized apps on their personal devices. This demand has given rise to the *iASPD* ecosystem: an underground chain that monetizes Apple's signing-provisioning-distribution pipeline, from certificate acquisition to revocation checks.

Fig. 2 depicts the *iASPD* ecosystem. Beyond the demand side (developers and end users), the ecosystem comprises several layers of actors with distinct profit models:

- **Promoters (Layer 4)** operate on social media and e-commerce platforms, redirecting customers to affiliated service frontends for referral fees.
- **Service frontends (Layer 3)** form the public-facing layer, absorbing demand and providing app-hosting and signing interfaces for a fee.
- **Intermediaries (Layer 2)** broker signing capacity: they acquire resources from upstream suppliers and resell to frontends or promoters, profiting from price differences.
- **Upstream suppliers (Layer 1)** hoard and maintain pools of Apple developer accounts (for ad hoc) and enterprise accounts (for in-house), often using leaked or fake identities, shell companies, or account trading, and sell signing certificates or account access to intermediaries.
- **Apple platform (Layer 0)** is the exploited legitimate infrastructure, including the Apple Developer Program (ad hoc), the Apple Developer Enterprise Program (in-house), runtime online verification (OCSP/PPQ), and certificate revocation.

Figure 2 also shows how demand propagates upward: developers and users drive the need for signing capacity, fueling a reseller market that abuses Apple's code-signing and provisioning materials. Apple retains control through certificate revocation, which temporarily disrupts the chain but has led to adaptive strategies among downstream actors.

We next detail the ecosystem from its two demand sources.

A. Developer Demand

Reaching out to service frontends. *App developers* seeking to distribute unauthorized apps (*e.g.*, malware, gambling, pirated content) (❶(a)) rarely manage signing themselves, as acquiring and maintaining Apple developer accounts is costly and technically demanding (see §II-A). Instead, they interact with *promoters* on social media or e-commerce platforms, who redirect them to affiliated *service frontends* (❷). These frontends act as underground app markets—IPA-signing and IPA-hosting sites offering customized signing and distribution.

Distribution and signing preferences. Service frontends procure signing resources from *intermediaries* (❸) according to the developer's intended distribution scale and target audience. For broad distribution of a limited number of apps, developers prefer *in-house distribution (enterprise signing)*: once signed, an app can be deployed without device constraints. Because enterprise certificates are scarce and heavily monitored, upstream actors typically retain them to avoid signing too many apps; intermediaries package this capability into *app-based resale* (charging per app). For narrower or targeted

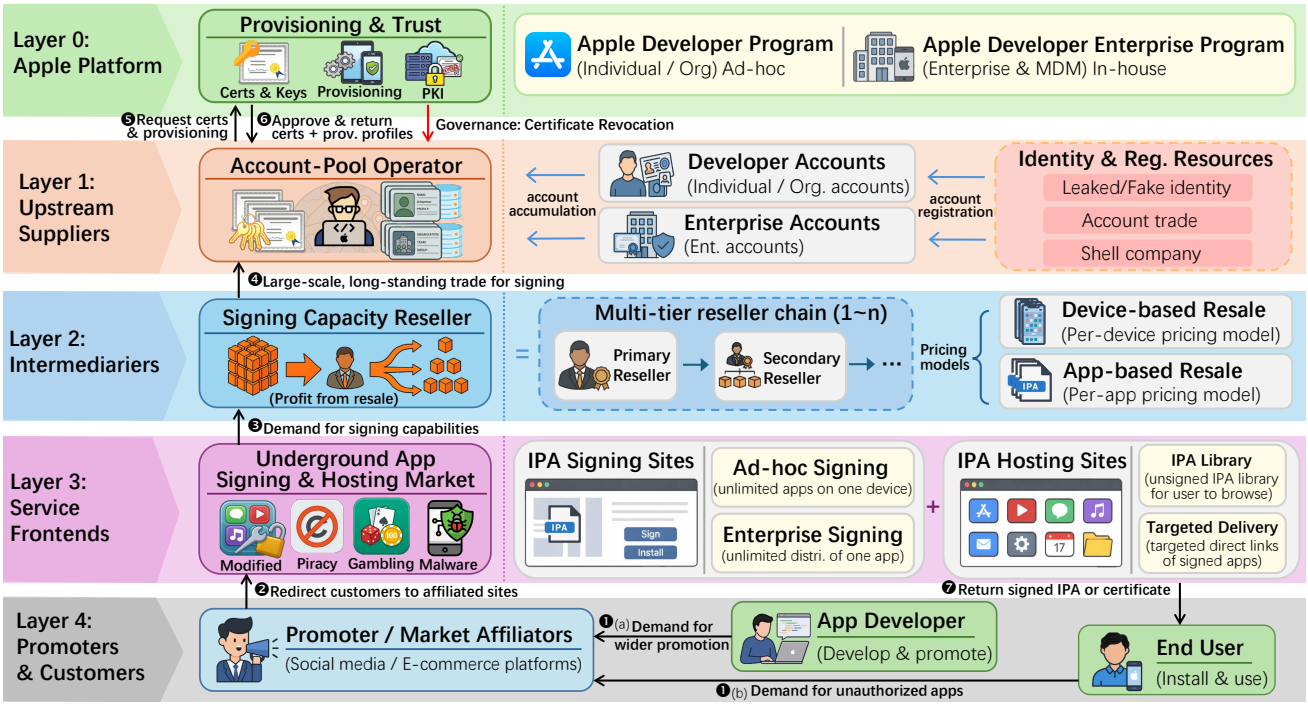


Fig. 2: Overview of the *iASPD* ecosystem. Customer demand (Layer 4) is absorbed by service frontends (Layer 3), brokered by intermediaries (Layer 2), satisfied by upstream account suppliers (Layer 1), and ultimately enabled by Apple's legitimate provisioning infrastructure (Layer 0).

distribution, developers may use *ad hoc signing*, which is tied to registered device UDIDs. The corresponding commercial model is *device-based resale* (charging per device), reflecting that once a device is registered, multiple apps can be installed without consuming additional device slots.

Reaching out to upstream suppliers. Demand for signing ultimately reaches *upstream suppliers* (❹), who control account pools (Apple Developer and Enterprise programs). These accounts are costly to obtain and sustain; suppliers acquire them via leaked or fake identities, account trading, or shell companies.

Existence of intermediaries. Because service frontends vary in demand scale, supply concentrates in a multi-tier reseller chain that supports large-scale, long-standing trade in signing capacity (❹). This structure allows upstream suppliers to profit sufficiently to maintain large pools, while frontends obtain only the capacity they need. It also buffers frontends from upstream volatility: when Apple revokes an account, disruption can be absorbed by replacing the affected resource higher in the chain or rerouting supply through alternative upstreams.

B. User Demand

Reaching out to service frontends. End users seeking unauthorized apps (*e.g.*, cracked games, emulators, mods) (❶(b)) enter the ecosystem via promoters on social media or e-commerce platforms, who redirect them to service frontends (❷) for affiliate commissions.

Per-device pricing model. Unlike developers, individual users typically install many apps on a few personal devices, making ad hoc signing the primary method. A user's device UDID must be registered, consuming a slot from a developer account (limited to 100 per account). The frontend translates user demand into demand for device slots, procured through intermediaries until reaching an account-pool operator. End users are charged per device slot, often with warranties (*e.g.*, monthly or annual), giving rise to a *per-device pricing model*.

Two delivery options. At the service frontend layer, IPA-hosting sites offer two delivery modes. In the *targeted-delivery* model, the site provides a direct download link for a specific app (supplied by developers who paid hosting fees). In the *IPA-library* model, the site acts as an unsigned IPA repository where users can search for apps. IPA-signing sites require users to submit their device UDID, redeem an activation code (often purchased from promoters), select an IPA, and receive a signed version. Together, hosting and signing complete the final delivery to the end user (❼).

Reaching out to upstream suppliers. The account-pool operator interfaces with Apple's platform to obtain signing materials (❺). The upstream returns the developer certificate (`.p12`) and device-bound provisioning profile (`.mobileprovision`) (❻). The IPA-signing site then code-signs the IPA and provides a download link to the user (❼). Many sites also provide users with the ad-hoc signing materials bound to their devices, enabling them to sign additional apps themselves.

C. Ecosystem Resilience to Apple's Governance

Apple curbs unauthorized distribution through usage monitoring, account bans, and certificate revocation. A revoked certificate normally renders apps non-functional unless users employ countermeasures, such as a custom DNS blocking OCSP verification requests to Apple [27], [28], potentially allowing the app to run for an undocumented period.

When revocation occurs, users can return to the same frontend for a replacement. Under the IPA-library model, this is a replacement request; under targeted delivery, the frontend may replace the signed package behind the download link. The request propagates upward through the reseller chain. The upstream side allocates a fresh developer account or certificate, returns new signing materials, and enables the frontend to re-sign the app. The user can then reinstall and continue using the newly signed unauthorized app.

Apple may impose a 30-day waiting period for ad hoc distribution when a device UDID previously linked to a banned account is registered under a new account. In-house distribution has no device registration, so replacement typically requires only a fresh enterprise certificate. From the user's perspective, Apple's governance manifests indirectly: revocation stops the app, but restoration is often handled through the frontend: either after the ad hoc waiting period or through enterprise signing. This resilience depends on the broader reseller chain and upstream account pools.

D. Summary

Both demand chains follow the same operational logic. Developers and users generate downstream demand, which promoters redirect to service frontends that provide app signing and hosting. Intermediaries turn this demand into resalable signing capacity through device-based or app-based pricing, while upstream suppliers maintain the account pools that meet it. The entire ecosystem rests on Apple's code-signing and provisioning infrastructure, with certificate revocation as the central governance constraint: one that can often be absorbed along the chain. By decoupling demand generation, service delivery, resale, and account maintenance, the *iASPD* ecosystem transforms Apple's own provisioning mechanisms into a resilient underground service.

IV. Data Collection

To answer our research questions and demystify the *iASPD* ecosystem, we use seized internal databases of illegitimate iOS signing and distribution operations, obtained through law enforcement collaboration. We supplement these with online conversations on social-media and e-commerce platforms with providers offering signing or hosting services, posing as potential customers or brokers. Below, we describe the seized databases and the conversation procedure.

A. Seized Data

Our primary data source comprises internal databases seized by law enforcement from underground iOS app distribution operations. Unlike external measurements that only see the user-facing surface, these databases reveal the operator-side view: registered users, promotions, pricing plans, apps, signing-request routing, certificates, and Apple developer accounts. They thus capture the full stack from service frontends to upstream suppliers. We group them by operational role.

Service frontends. These are the main entry points for app developers and end users, and the entities with which promoters have referral agreements. Their records cover frontend services (signing and hosting) as well as customer and promoter activities. Specifically, they contain public pricing plans for ad-hoc and enterprise signing; metadata and download links for distributed apps; and internal records such as download/signing requests, registered accounts (developers, end users, promoters) with PII (*e.g.*, email, phone), payment histories (amount and time), and affiliate histories (commission, withdrawals). These records support our analysis of customer profiles, promoter monetization, and app categorization.

Intermediaries. Intermediaries buy signing capacity from upstream suppliers and sell it to service frontends, profiting from the price difference. Their records, including signing requests, routed certificates, reseller pricing plans, and payment histories, enable analysis of forwarding strategies, multi-tier resale networks, and monetization.

Upstream suppliers. These are the actual Apple developer account holders who manage signing materials. Their records contain developer account credentials, certificates (issue, expiry, revocation dates, names), and provisioning profiles (App IDs, capabilities, device UDIDs for ad-hoc). These records help analyze account sources, scale, activity, and how the ecosystem sustains persistent abuse through replacement or relocation of signing materials.

B. Social Media and E-commerce Platforms

While seized databases offer operator-side insights, they lack visibility into how Apple developer accounts are sourced and maintained, and how promoters attract customers outside service frontends. To fill these gaps, we conducted online conversations on social-media and e-commerce platforms with highly ranked providers offering signing or hosting capabilities. We approached them as potential customers or brokers, depending on whether the interaction targeted customer-facing promotion or upstream supply.

E-commerce. To study promoter reach, we searched for keywords like "*Apple/iOS/IPA sign/cert*" on popular e-commerce platforms, including *JD.com*, *Taobao*, and *Goofish* (a second-hand platform). These platforms often list standalone pricing and promotional materials not found in our seized databases. We collected merchants selling signing services and recorded their pricing plans, service descriptions, and user reviews.

Social media. To investigate upstream supply, we searched the same keywords on *Telegram*, *QQ*, and the open web. Because these channels rarely publish complete pricing plans before direct contact, we therefore conducted online conversations with identified providers. Many claimed to be upstream suppliers targeting bulk signing demand. Posing as customers,

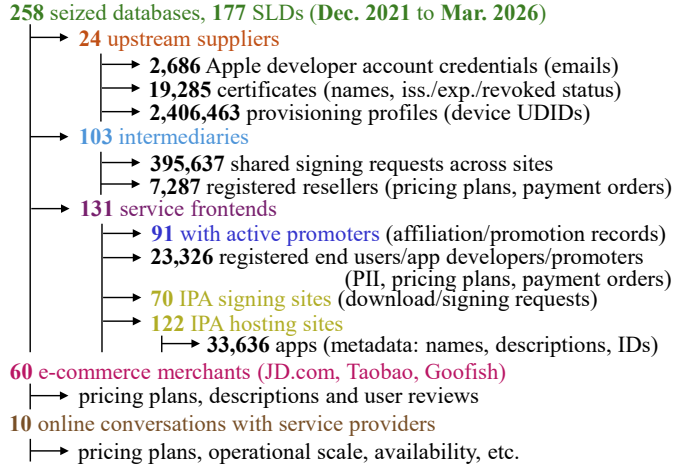


Fig. 3: Summary of collected data from seized databases, social-media, and e-commerce platforms.

TABLE II: Top five platforms by revenue (RMB), with active period, user count, and request count; the last row aggregates all 258 platforms.

Platform	Role	Revenue(¥)	Active Period	#Users	#Requests
SPEED	*upstream**	4,074,012	2023/07 - 2026/01	2,900	974,839
SUPERB	*upstream**	2,784,607	2023/11 - 2026/01	448	769,943
SHRIMP	*frontend*	2,751,059	2021/12 - 2025/12	15,237	1,284,910
EASY	*intermediary*	668,753	2024/07 - 2026/03	1,689	93,311
LITTLE	*intermediary*	507,307	2023/11 - 2026/01	345	55,741
All-258	**—**	**12,209,551**	**2021/12 - 2026/03**	**31,955**	**3,490,438**

*upstream**: also exhibit *intermediary* characteristics.

we obtained detailed pricing plans; posing as brokers, we collected information on their operational scale, signing material availability, and claimed sources of Apple developer accounts.

C. Data Processing

To enable unified analysis, we manually inspected the heterogeneous seized databases, mapped equivalent fields, and linked records into a unified data model. After removing invalid and obsolete entries, we constructed a dataset with eight tables: developer accounts, certificates, provisioning profiles, signing requests[1], registered accounts, payment orders, pricing plans, and apps. Tables XI and XII in Appendix A detail the fields and relationships of each table. The databases were seized between Dec. 2025 and Mar. 2026. All online checks, including revocation status, were performed in Apr. 2026.

D. Data Summary

Fig. 3 summarizes our data. We obtained 258 seized backend databases, corresponding to 177 second-level domains (SLDs) that likely represent distinct operating entities.

- **Upstream:** 24 databases contain 2,686 Apple developer account credentials (mostly email addresses), 19,285 certificates (with names, issue/expiry/revocation status), and 2,406,463 mobile provisioning profiles (device UDIDs).
- **Intermediary:** 103 databases contain 395,637 signing requests shared across at least two intermediaries, plus 7,287 registered resellers with pricing plans and payment orders.
- **Service frontend:** 131 databases contain 23,326 registered accounts (end users, developers, promoters) with PII, pricing plans, and payment orders. Among these, 70 offer signing (with signing request records), 122 offer hosting (with 33,636 app metadata records including app names, descriptions, Bundle IDs), and 61 offer both.

Table II highlights several representative platforms. The largest upstream supplier, *SPEED* (active 2023–2026 in the seized data), generated 4,074,012 RMB with 2,900 users and 974,839 signing requests. The largest intermediary, *EASY* (2024–2026), earned 668,753 RMB with 1,689 users and 93,311 signing requests. The largest frontend, *SHRIMP* (2021–2025), earned 2,751,059 RMB with 15,237 users and 1,284,910 app download requests. The end dates correspond to the time of database seizure. While individual revenues may appear modest, our recorded revenue is incomplete: for instance, *SUPERB* has been active since Nov. 2023, but its payment records begin in Jan. 2025, likely due to operator-side manual truncation. Overall, the seized data includes 24 upstream suppliers, 103 intermediaries, and 131 frontends, with a total accumulated revenue of 12,209,551 RMB, indicating a substantial underground market size in the wild.

Additionally, we sampled 60 merchants on JD.com, Taobao, and Goofish, recording their pricing plans, service descriptions, and user reviews to analyze promoter activity. Online conversations with 10 signing/hosting providers (all ultimately via Telegram) yielded detailed pricing for bulk signing/hosting, along with information on operational scale, resource availability, and claimed sources of developer accounts.

V. Data Analysis

Using the seized databases, e-commerce records, and online conversations, we analyze the *iASPD* ecosystem from three angles: the demand-facing layer (customers, promoters, service frontends), intermediary resale chains, and upstream suppliers.

A. Customers, Promoters, and Service Frontends

We first characterize customer profiles, promoter profit models, and service-frontend records (affiliation, pricing plans, distributed apps).

1) Customers: Customer demand stems from two groups: developers seeking alternative distribution channels, and end users seeking apps that are unavailable, restricted, or modified. Service frontends convert this demand into signing and hosting services, often through promoter redirection.

Developers. Developers turn to frontends to reduce distribution costs and bypass operational constraints (Table I) or to distribute apps unlikely to pass App Store review. Their ages have a central 90% interval of 23–49; a few are over 55 or under 18. Some affiliated companies are online-identified "shell companies". Developer demand is skewed: 85% distribute fewer than five apps (half only one), and only a small fraction (<5%) distribute more than ten apps (maximum 129).

[1]A signing request denotes a successful platform-side request for which the provisioning profile and certificate are finally available. Requests resolving to the same signing materials are deduplicated per platform.

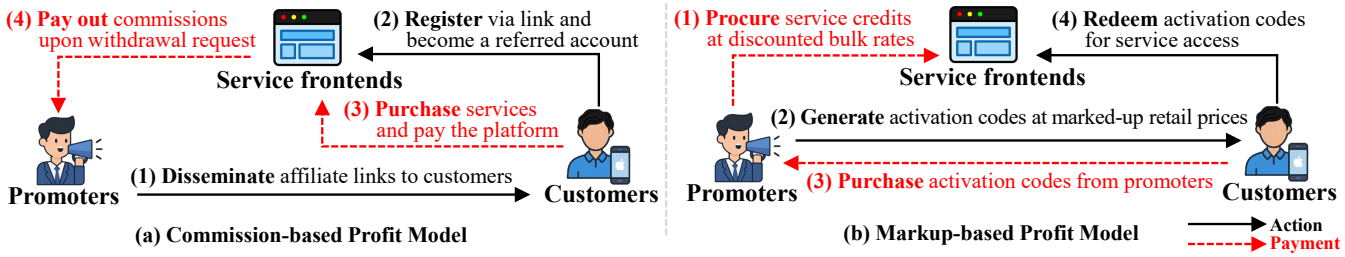


Fig. 4: Two profit models for promoters: commission-based and markup-based.

Prepaid fees for signing and hosting vary substantially: the largest prepaid amount for a single app reaches 34,848 RMB, while the central 90% ranges from 10 RMB to 1,000 RMB per app, with a median of 60 RMB and a mean of 200 RMB.

End users. End users seek apps that are paid, restricted, or prohibited in the App Store. Lacking technical expertise, they rely on frontends, via promoters, that provide signing and hosting services. They favor cracked games, unlocked premium features, and modified social apps, and generally prefer free or low-cost options [14]. This preference enables developers or operators who prepay fees and distribute direct download links (the *targeted-delivery* model) to circulate fraud, scam, and malware apps through the same infrastructure.

2) Promoters: Promoters attract demand on social-media and e-commerce platforms and profit via two models: *commission-based* and *markup-based*, as illustrated in Fig. 4.

(a) Commission-based model. Some service frontends implement an affiliate mechanism. The process typically unfolds as follows: (i) a promoter shares an affiliate link via social media or other channels; (ii) when a new customer registers through that link, their account becomes permanently linked to the promoter; (iii) the customer purchases services and pays the frontend; and (iv) the frontend pays the promoter a commission on each such payment.

(b) Markup-based model. Promoters can also profit from price differences, a model common on e-commerce platforms. The process typically proceeds as follows: (i) promoters purchase service credits (account balances) from frontends at discounted bulk rates; (ii) they generate activation codes and sell them on e-commerce platforms at marked-up retail prices; (iii) customers buy these codes from promoters at prices above the promoters' original cost; and (iv) customers redeem the codes on the frontend to obtain the desired service.

3) Service Frontends: Service frontends are the main operational entry point for both customers and promoters, making their records a rich source of insight into the demand-facing layer. Affiliation records show how commission-based promoter monetization works in practice; pricing plans reveal how frontends set prices for signing and hosting services under different technical and commercial constraints; and distributed app records expose the types of apps flowing through this infrastructure and the risks they introduce.

(3a) Affiliation

Of the 131 service frontends, 91 support promoter affiliation. The leading frontend, *SHRIMP*, offers commission rates between 10% and 20%, varying by promoter. On *SHRIMP*, 24 distinct promoters with a total of 417 referred accounts (17.4 per promoter) made 260 commission withdrawals totaling 102,262 RMB from Oct. 2022 to Dec. 2025. The top four promoters received 80.2% of this amount (82,014 RMB), indicating that successful promoters can earn substantial revenue.

TABLE III: Pricing plans of service frontends and e-commerce merchants selling activation codes, across device types and warranty periods. Prices are reported in RMB per device slot.

		Avg. Price w/ Warranty Duration			
Plan	**Device**	**Zero**	**40-day**	**180-day**	**330-day**
Instant	iPhone	11.0	13.0	18.2	28.2
Instant	iPad	5.0	4.8	5.6	7.7
Activ.-code	iPhone	—	16.0	35.1	49.7
Activ.-code	iPad	—	4.2	8.1	13.1
Reservation	iPhone	6.2	8.4	15.1	20.2

(3b) Pricing Plans

We identify two pricing models: *per-device* (for ad hoc) and *per-app* (enterprise).

Per-device pricing. Under ad-hoc, each developer account is limited to 100 device slots per year, making device slots a scarce resource sold to end users. Frontends offer plans with different warranty periods (*i.e.*, guaranteed free replacement if Apple revokes a certificate). Table III summarizes the pricing plans across 131 frontends and 60 e-commerce merchants.

We describe the three most representative plans as follows. (i) *Instant*: users receive signing certificates almost immediately or after minimal delay, resulting in a higher price. (ii) *Reservation*: users may wait up to two days for certificates to become effective, leading to a lower price. (iii) *Activation code*: codes sold by promoters on e-commerce platforms. These codes are redeemable on service frontends for *Instant* plans (same device type and warranty period) and are listed here for direct comparison.

Table III shows that iPhone slots are priced higher than iPad slots. Prices also rise with longer warranty periods: extending from no warranty to 40-day, 180-day, or 330-day guarantees substantially increases the price, reaching approximately 2 to 4 times the no-warranty price for some plans. Activation-code prices are roughly 2× higher than *Instant*, indicating a large profit margin for promoters who sell these prepaid codes.

Per-app pricing. This model serves developers who need wide distribution via in-house signing. Table IV summarizes

TABLE IV: Pricing plans in USDT (Tether USD) for per-app distribution from five providers (A–E).

Fee Type	Price_A	Price_B	Price_C	Price_D	Price_E
Normal signing (monthly)	350	200	120	180	180
Dedicated signing (monthly)	850	—	600	750	600
Hosting per download	0.44	0.30	—	0.59	0.44
Minimum daily spend	—	3–300	—	5.9	4.4–44

pricing plans from five signing/hosting service providers (denoted A–E). A *normal signing* plan, for signing many apps, costs 120–350 Tether USD (USDT) per month per app. A *dedicated signing* plan, signing fewer apps to avoid Apple's attention, costs 600–850 USDT per month per app (provider B does not offer this plan). Most providers charge a *hosting fee per download* (0.30–0.59 USDT), and some impose a minimum daily spending requirement (up to 300 USDT). These requirements indicate substantial financial commitments for distribution. Notably, all providers offering enterprise signing requested cryptocurrency payments. Some also required a 10% price increase for fraud, scams, or malware apps.

(3c) Underground Apps

To identify the types of apps distributed through this ecosystem, we analyzed 33,636 iOS app records from the seized databases. Among these, 30,618 were unsigned; the remainder included 606 ad-hoc-signed and 2,412 enterprise-signed apps. Additionally, the same platforms hosted 931 WebClips and 5,637 Android apps. The following analysis focuses exclusively on native iOS apps.

Abusive category. We deduplicated app names and Bundle IDs, obtaining 22,259 unique native iOS apps. Each app was assigned to a single abusive category based on its dominant advertised abusive behavior, determined from app names, Bundle IDs, descriptions, and keywords. For example, an app was labeled *Game Cheat* if its metadata advertised gameplay manipulation (*e.g.*, unlimited currency, god mode, one-hit kill, damage multipliers, unlocked items, or built-in mod menus). An app was labeled *Premium Access* when the primary abusive behavior was bypassing paid entitlements, subscriptions, advertisements, or VIP tiers (*e.g.*, *Spotify* variants claiming premium access or *Meitu* variants advertising SVIP unlocks). An app was labeled *Modified Variants* when it mainly altered a mainstream client beyond unlocking a paid feature, such as *WeChat*, *WhatsApp*, *TikTok*, or *Xiaohongshu* variants featuring multi-open support, plug-in injection, anti-recall, watermark removal, camera spoofing, or theme modification.

Overall, 18,555 apps (83.4%) were categorized as abusive; the remainder lacked sufficient evidence for classification. Table V breaks down these 18,555 apps by category. The top three categories, *Game Cheat* (7,609; 41.0%), *Premium Access* (6,646; 35.8%), and *Modified Variants* (3,345; 18.0%), together account for 94.9% of all categorized apps, reflecting strong demand for cracked games, unlocked premium features, and modified mainstream apps.

General category of base apps. We normalized app names to recover their base app families (*e.g.*, "TikTok-BH++" → "TikTok") and queried the iTunes Search API to check official App Store presence. From the 22,259 unique native-iOS apps, we identified 16,844 distinct base apps. The largest base families are widely used mainstream apps: *WeChat* (1,346 variants), *TikTok* (149), *QQ* (65), *Telegram* (53), *Instagram* (52), *Bilibili* (49), and *Meitu* (35). We matched 6,886 base apps (40.9%) to App Store records; Table VI shows their general category distribution. Among matched apps, 440 (6.4%) are paid App Store apps (*e.g.*, *GoodReader*, *Minecraft*). Thus, underground app markets redistribute not only standalone abusive software but also mainstream App Store apps, including paid ones, through unofficial channels.

Static analysis. We extracted 2,238 enterprise-signed IPA files from frontend databases and submitted them to VirusTotal for malware detection. Among these, 1,117 apps (49.9%) received at least one malicious flag, and 595 (26.6%) were flagged by ten or more antivirus engines. Their upload dates span from Jan. 2023 to Oct. 2025. The dominant malicious

TABLE V: Abusive category distribution among 18,555 labeled native-iOS apps.

Abusive Category	#Apps	Representative Features
Game Cheat	7,609 (41.0%)	God mode; one-hit kill; infinite coins
Premium Access	6,646 (35.8%)	VIP/pro unlock; ad removal
Modified Variants	3,345 (18.0%)	Multi-open; anti-recall; camera spoof
Gambling	484 (2.6%)	Poker/mahjon; coin-pusher
Media Piracy	108 (0.6%)	Comic/novel reader; ad-free streaming
Adult Content	40 (0.2%)	Adult dating; paid voice/video dating
VPN / Proxy	100 (0.5%)	VPN; accelerator; geo-bypass
Unlicensed Finance	131 (0.7%)	Crypto wallet; loans; pyramid scheme
Sideloading	52 (0.3%)	Signing tool; IPA installer
Jailbreaking Tool	40 (0.2%)	Dopamine/unc0ver; CVE exploit

TABLE VI: General category distribution among 6,886 matched App Store apps.

General Category	#Apps	Representative Apps
Games	2,271	Plants vs. Zombies; PUBG MOBILE; Minecraft
Social Networking	1,362	WeChat; Telegram; QQ
Photo & Video	734	Instagram; YouTube; BeautyCam
Utilities	571	iBox; FileBrowser; Loon
Entertainment	442	TikTok; bilibili; Infuse
Productivity	268	Notability; ChatGPT; WPS Office
Education	191	Duolingo; Busuu; Mondly
Health & Fitness	135	Pillow; MyFitnessPal; Keep
Music	131	Spotify; YouTube Music; GarageBand
Graphics & Design	117	Procreate Pocket; ibis Paint X; Adobe Express
Business	97	TIM; WhatsApp Business; DingTalk
Lifestyle	83	Alipay; Pinterest; Planner 5D
Book	67	Tomato Novel; Qidian Reader; KyBook
Finance	60	Binance; DayCost Pro; Buddy
Reference	58	Eudic; Google Translate; ReverseVoice
Shopping	56	Taobao; Xianyu; Pinduoduo
Weather	45	Caiyun Weather; Moji Weather
Navigation	40	AMap; Baidu Maps; Guru Maps
News	36	X; Toutiao; Reddit
Developer Tools	32	App Store Connect; Textastic; GitHub
Sports	30	FotMob; NBA Live; Tencent Sports
Travel	25	Flightradar24; Tripsy; FlightAware
Food & Drink	17	Ele.me; Meituan Waimai; Luckin Coffee
Medical	14	SnoreLab; 3Dbody; Clarity
Newspapers	3	Caixin; IT Home
Stickers	1	Anime Now

family is `SparkKitty/SpyAgent` [29], accounting for 945 (84.6%) of the malicious apps. Other major detections include 138 (12.4%) labeled as `unclassified trojan`, 20 as `TinyV/Tiniv` [30], and 10 as `Jailbreak/PrivEsc`. Manual validation identified three VirusTotal false positives attributable to advertisement SDKs and two false negatives that exhibited customized command-and-control behavior.

Sensitive privacy declarations are common: 883 (79.1%) request microphone access, 823 (73.7%) photo-library access, 742 (66.4%) location access, 724 (64.8%) camera access, and 189 (16.9%) contacts access. The `SparkKitty/SpyAgent` samples are created either through simple dylib injection or complete app rebuilding; they commonly request microphone, photo-library, or other broad permissions, then monitor and exfiltrate sensitive data.

These malicious apps predominantly fall into *Modified Variants*, *Gambling*, and *Unlicensed Finance*. Apps lacking overt abusive metadata often disguise themselves as private-server games, chat apps, business or productivity tools, or live/media apps to attract users. Although malicious apps represent only 49.9% of signed IPA files, they account for 64.2% (720,223) of recorded downloads among these 2,238 apps.

We further examined the development frameworks used by the malicious apps. Among the 1,117 malicious apps, 63.7% (711) are built with cross-platform frameworks: 437 (39.1%) use *cocos2d-x* (a cross-platform game engine), and 274 (24.5%) use others including *DCloud*, *Unity*, *Flutter*, *React*, and *WebView*. This prevalence suggests that many malicious apps are designed for reuse beyond iOS. Further analysis shows that 303 (42.6%) malicious apps share the same app name and identifier in both Android and iOS records, supporting the extensive use of cross-platform frameworks.

B. Intermediaries: Multi-tier Reseller Chains

Intermediaries resell signing capacity: they acquire signing materials from upstream suppliers and resell them to service frontends, using a markup-based profit model similar to that of promoters (Fig. 4(b)). In addition, they often organize into multi-tier reseller chains that route signing requests across multiple intermediaries.

1) Signing Requests Routing: When a service frontend receives a customer signing request, it may forward the request through one or more intermediaries before an upstream supplier resolves it by allocating the appropriate developer account and returning the signing materials downstream.

An intermediary can have multiple *direct higher-tier intermediaries (DHIs)*: the next-hop intermediaries to which it directly forwards requests. Consequently, the path from the request issuer (service frontend) to the resolver (upstream supplier) is dynamic and flexible. This flexibility underpins the ecosystem's resilience: when Apple revokes a certificate, subsequent requests can be rerouted through alternative reseller chains to obtain fresh signing materials. We next analyze the request-forwarding strategies observed in the seized records.

Request-forwarding strategies. Of the 103 intermediaries, 31 have *DHI* configuration information available in the seized data. For these, we identify three commonly used request-forwarding strategies, illustrated in Fig. 5.

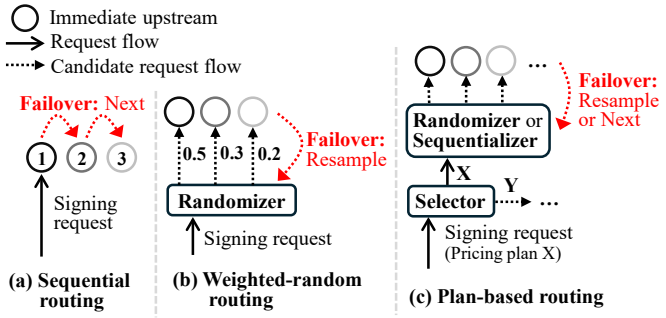


Fig. 5: Three request-forwarding strategies adopted by intermediaries: sequential, weighted-random, and plan-based.

(a) Sequential routing. A signing request is forwarded through a fixed ordered list of DHIs. The request is routed to a DHI only when the preceding DHI fails. Twenty-four of the 31 intermediaries employ this strategy, with an average of 1.2 configured DHIs. These intermediaries typically reside at lower tiers of the reseller chain, closer to service frontends.

(b) Weighted-random routing. This strategy is used by intermediaries with multiple comparable DHIs. Each DHI is assigned a predefined weight; a higher weight increases the probability of selection. If the selected DHI fails, the intermediary resamples from the remaining DHIs using the same weights after excluding failed candidates. Three of the 31 intermediaries adopt this strategy, and all have at least three effective DHIs configured.

(c) Plan-based routing. This strategy selects DHIs according to attributes of the incoming signing request, such as warranty period or device type. It complements the previous strategies by adding a selector before the routing decision: only DHIs matching the request plan are passed to the sequential or weighted-random router.

For example, *SPEED* has nine DHIs. It sends all zero-warranty requests to a specific DHI, samples 40-day warranty requests across two DHIs at a 7:3 weight ratio, and assigns all 180-day and 330-day warranty requests to another dedicated DHI. Notably, some resellers under *SPEED* are configured with fresh DHIs and custom weights, indicating support for reseller-specific routing policies.

2) Intermediary Hierarchies: Reconstructing the full reseller network is challenging, as most *DHI*s identified in our datasets are not captured in the seized databases. We therefore rely on 395,637 shared signing requests that record the same certificate serial number across at least two intermediaries to infer relationships among them. Such sharing likely indicates direct hierarchical connections, meaning the same signing material passed through multiple intermediaries.

We infer hierarchy based on two simple intuitions. Let A, B, and C be three intermediaries. First, if many of A's requests also appear in B, but only a small portion of B's requests appear in A, then B is likely a direct higher-tier intermediary of

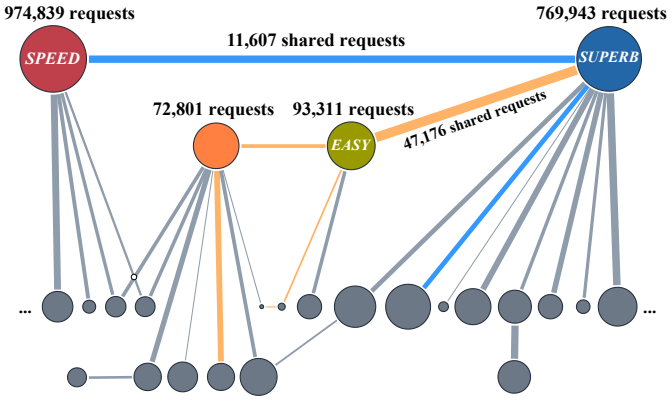


Fig. 6: Illustration of the reconstructed reseller network.

TABLE VII: Intermediary count, total and shared request counts, and recorded revenue (RMB) across four reconstructed reseller tiers and a standalone subset among 74 intermediaries.

Tier	#Inter.	Total #Reqs	Shared #Reqs	Revenue(RMB)
Tier-0	2	1,744,782 (80.0%)	196,748 (49.7%)	6,858,619 (72.6%)
Tier-1	25	319,978 (14.7%)	169,874 (42.9%)	1,891,816 (20.0%)
Tier-2	11	26,109 (1.2%)	18,552 (4.7%)	50,030 (0.5%)
Tier-3	3	3,383 (0.2%)	1,825 (0.5%)	1,049 (0.0%)
Standalone	33	86,590 (4.0%)	8,638 (2.2%)	649,109 (6.9%)
Total	**74**	**2,180,842 (100%)**	**395,637 (100%)**	**9,450,622 (100%)**

A. Second, larger entities with more requests or higher revenue are more likely to be higher-tier. Accordingly, we designate the two largest entities (*SPEED* and *SUPERB*) as tier-0 intermediaries, and then assign the remaining connected intermediaries to tier-1, tier-2, or tier-3 based on their hierarchical distance from tier-0. Fig. 6 illustrates a simplified view of the reconstructed reseller network. Nodes represent intermediaries, and edges represent shared requests; an edge indicates an inferred *direct higher-tier intermediary* relationship. Both node and edge sizes are log-scaled according to request volume and shared-request volume, respectively. The blue edges, read from bottom to top, show one recovered reseller chain of length 3. The orange edges show two additional recovered chains of length 4.

Table VII presents the reconstructed reseller network. Among the 74 intermediaries, 41 (55.4%) are assigned to a tier; the remaining 33 are standalone, not connected to the tiered structure. The network is highly concentrated. The two tier-0 intermediaries account for 80.0% of total requests (1,744,782) and 72.6% of revenue (6,858,619 RMB). Tier-1 comprises 25 intermediaries (14.7% requests, 20.0% revenue). Tier-2 and tier-3 contribute smaller volumes; the three tier-3 intermediaries generated only 3,383 requests, yet their presence shows that some reseller chains can extend across at least three intermediary tiers before reaching higher-tier suppliers.

Overall, the reseller chains operate as an inverted funnel: many lower-tier intermediaries absorb demand from service frontends, while requests and revenue concentrate toward a small number of higher-tier hubs. Our data clearly captures this concentration at tier-0 and tier-1.

We further linked these tiers to 7,287 registered resellers, each acting as a direct lower-tier intermediary. The top 100 resellers (1.4%) accounted for 1,405,924 signing requests (64.5% of total) and 5,677,065 RMB in revenue (60.1%). In contrast, most resellers were low-volume: 78.0% generated $<$10 requests and contributed only 1.4% of total revenue.

3) Profit Margins: Two major intermediaries, *SPEED* and *EASY*, recorded detailed pricing information in their signing-request logs: the advertised list price, the actual transaction price, and the procurement cost. For each per-device plan, we compute the profit per request as transaction price minus procurement cost. Table VIII summarizes these figures, including only requests with complete records.

For *SPEED*, procurement costs range from 8.4 RMB (0-day) to 14.9 RMB (330-day). Long-warranty plans (180-day and 330-day) are sold at roughly twice their cost, generating per-request profits of 14.0 RMB and 17.6 RMB respectively — more than ten times the profit of short-warranty plans (0-day: 1.0 RMB). *EASY* exhibits a similar pattern, with 180-day and 330-day profits reaching 18.3 RMB and 27.3 RMB. On both platforms, high-volume resellers (over 5,000 requests) obtain discounted wholesale prices for long-warranty plans (20 RMB and 26 RMB), indicating volume-based price discrimination. Although short-warranty plans have thin per-request profits, their high request volumes (351,719 for *SPEED* 0-day) drive substantial aggregate profit. Together, *SPEED* and *EASY* generated 1,102,052 RMB in total profit across these plans. Since only requests with complete pricing data are considered, this figure likely underestimates the true profit.

4) Dynamics Modeling: The observed routing strategies can be formalized by a simple forwarding model. For an intermediary i, let $x = (d, w)$ denote a request class, $V_i(x)$ its volume, and $F_i(x)$ the eligible direct higher-tier intermediaries. Routing a request through $j \in F_i(x)$ yields per-request margin $M_j(x) = p_i(x) - c_j(x) - \ell_j(x)$, representing sale price, procurement cost, and expected loss due to warranty or revocation, respectively. If $\rho_j(x)$ denotes the probability that j ultimately resolves the request under i's routing policy, then the expected aggregate profit is $\Pi_i = \sum_x V_i(x) \sum_{j \in F_i(x)} \rho_j(x) M_j(x)$. Each intermediary configures its routing policy to balance profit and service availability. These local decisions shape a dynamic supply chain whose rerouting flexibility underpins resilience after revocation.

C. Upstream Suppliers: Certificates and Accounts

Upstream suppliers maintain pools of Apple developer accounts and fulfill signing requests from intermediaries. They supply signing materials—certificates and provisioning profiles—to lower layers of the *iASPD* ecosystem. We analyze developer-account abuse for enterprise and ad-hoc signing.

1) Enterprise Signing: Enterprise certificates are scarce and typically held by upstream suppliers, so our dataset contains limited direct information on enterprise developer accounts. We therefore focus on the 2,238 enterprise-signed IPA files distributed via service frontends (§V-A(c)). From these, we extracted 518 distinct provisioning profiles and identified 306

TABLE VIII: Pricing plans, request counts, and per-request profits (RMB) for two intermediaries. *List price* is the advertised price; *Sale price* is the actual price paid; *Procurement cost* is the intermediary's cost. Profit = Sale price − Procurement cost.

		Avg. Price			Profit	
Pricing Plan	**#Reqs**	**List Price**	**Sale Price**	**Procure. Cost**	**Avg. Profit**	**Total Profit**
SPEED (0-day)	351,719	15.0	9.4	8.4	1.0	351,719
SPEED (40-day)	227,185	20.0	9.4	8.5	0.9	188,564
SPEED (180-day)	5,522	35.0	28.4	14.4	14.0	77,308
SPEED (330-day)	9,989	40.0	32.5	14.9	17.6	175,707
EASY (0-day)	71,449	16.0	9.9	8.6	1.3	91,455
EASY (40-day)	43,463	18.0	10.3	7.5	2.8	122,131
EASY (180-day)	1,517	35.0	26.6	8.3	18.3	27,701
EASY (330-day)	2,475	45.0	35.0	7.8	27.3	67,469
Total	**713,319**	**17.53**	**10.10**	**8.56**	**1.54**	**1,102,052**

TABLE IX: Top 20 enterprise certificates by number of signed apps and maximum observed app lifespan.

Enterprise Certificate (Name)	#Apps	Max Lifespan (d)
(1) HDFC Bank Limited	95	64.3
(2) Sunshine Insurance Group Co., Ltd.	93	51.1
(3) Etisalat – Emirates Telecommunications Corporation	87	96.5
(4) China CITIC Bank Corporation Limited	85	50.3
(5) GAC TOYOTA MOTOR CO.,LTD	75	42.7
(6) Wasu Media & Network Co., Ltd.	60	16.7
(7) SINOPEC SABIC Tianjin Petrochemical Co. Ltd.	59	61.0
(8) Kotak Mahindra Bank Ltd	56	69.2
(9) China Telecommunications Corporation	49	54.1
(10) 58 Daojia Life Service Co., Ltd.	41	86.9
(11) Shenzhen SXT Software Co.,Ltd.	40	35.6
(12) China Mobile Group Shandong Co., Ltd.	37	75.0
(13) SIE Consulting Co.,Ltd.	34	51.2
(14) PNB MetLife India Insurance Company Limited.	32	28.1
(15) BMW Brilliance Automotive Ltd.	32	25.3
(16) FPT Software Company Limited	31	27.0
(17) Kotak Mahindra Life Insurance Company Ltd.	28	36.6
(18) Jiajiayue Group Co.,Ltd	28	20.1
(19) MISFIT WEARABLES CORPORATION (Ent)	28	116.8
(20) LuZhou LaoJiao Co.,Ltd.	25	43.3

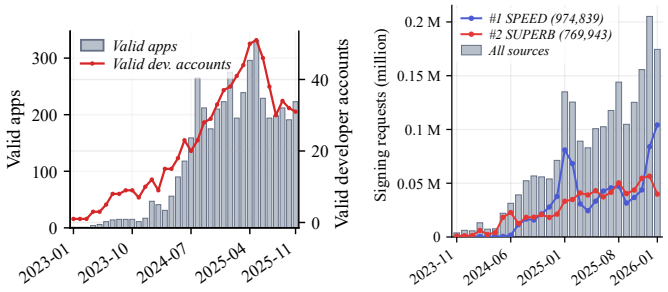
associated enterprise certificates, corresponding to 269 unique enterprise names. Combining this evidence with online conversations, we characterize the scale, activity, and persistence of enterprise-signing abuse.

(1a) Scale and Activity

Table IX lists the top 20 enterprise certificates by signed-app count and the maximum observed lifespan of apps signed with each certificate. An app's *lifespan* is the cumulative number of days it remained valid after signing before its signing materials expired or were revoked. The enterprise names span banking, telecommunications, software, and automotive sectors. Although many apps share the same certificate, most were distributed by different developers, indicating widespread certificate reuse or abuse. Notably, among the 1,015 apps (45.3%) signed using these top-20 certificates, 508 (50.0%) were detected as malicious (§V-A3(c)).

Public leaks. Twelve of the top-20 certificates appeared in public leaks where signing materials were freely shared in online communities [31], [32]. These leaks typically occurred one to three months before the associated provisioning profiles expired, suggesting that late-stage disclosure may attract demand shortly before the materials become unusable.

Activity. Fig. 7a shows monthly numbers of valid signed apps and unrevoked developer accounts. Both metrics increased



(a) Enterprise-signing activity **(b)** Ad-hoc-signing activity

Fig. 7: Temporal activity patterns: enterprise-signing measured by valid signed apps (2,238 enterprise-signed apps); ad-hoc-signing measured by request counts across all 258 platforms.

over time, peaking in April 2025. They declined slightly in the fourth quarter of 2025, possibly due to stronger Apple enforcement, but the number of valid signed apps remained substantial. Moreover, larger providers not covered by our seizure disclosed in online conversations that they distribute at least 10,000 enterprise-signed apps per month.

(1b) Persistence of Abuse

Validity period. Table IX also reports the maximum observed lifespan of apps signed with each certificate. For example, certificate #6 (Table IX) had a maximum lifespan of only 16.7 days, whereas #19 reached 116.8 days. By April 2026, all top-20 certificates had expired or been revoked. Across all 306 distinct enterprise certificates, only 20 (6.5%) remained valid, and they signed 29 underground apps. Among the 2,238 enterprise-signed apps, the average lifespan was 31 days (central 90%: 1–117 days). Nevertheless, 158 apps (7.1%) remained valid for more than three months, and some for up to one year, indicating that *certain enterprise-signed apps can maintain relatively long availability without promptly triggering Apple's enforcement.*

Certificate replacement. Abuse often persists through certificate replacement: an app becomes functional again after being re-signed with a new unrevoked certificate. Among the 2,238 enterprise-signed apps, 129 distinct apps (5.8%) were signed with multiple certificates. These apps had an average lifespan of 71 days (central 90%: 25–233 days, maximum 418 days), substantially extending their overall availability.

We also observed repeated abuse of the same developer accounts. For instance, six certificates issued by `Sunshine Insurance Group Co., Ltd.` were used to sign underground apps. The first certificate was issued in Dec. 2022 and revoked in Sep. 2023. Around revocation, a new certificate under the same account was issued. This pattern repeated six times within two years. As the number of apps signed by each certificate grew from one to five and eventually to 50, the certificate lifespan shortened from about half a year to a few weeks; by Sep. 2024, the latest certificate remained valid for only two weeks before revocation. More broadly, 29 of 269 enterprise developer accounts (10.8%) had multiple certificates in our records, indicating repeated abuse.

Account source. In online conversations, large providers disclosed that many enterprise certificates are leaked by insiders and sold at high prices, typically ranging from 200,000 to 2,000,000 RMB—a reflection of their scarcity. Despite these high acquisition costs, providers can still operate profitably at scale given their claimed distribution volume ($\geq$10,000 apps per month) and per-app fees (Table IV), underscoring the strong financial incentives behind enterprise-signing abuse.

2) Ad-hoc Signing: Ad-hoc signing relies on individual developer accounts subject to a 100-device limit. The ecosystem scales by pooling many such accounts and routing requests across them. We analyze 2,686 developer-account credentials, 19,285 certificates, and 2,406,463 provisioning profiles from 24 upstream suppliers, cross-referenced with signing-request logs and online provider conversations. We examine ad-hoc signing from three perspectives: scale and activity, persistence of abuse, and account sources.

(2a) Scale and Activity

Among 2,406,463 provisioning profiles, 2,405,833 (99.9%) are ad-hoc, corresponding to 1,710,310 unique device UDIDs. iPhones constitute the majority (92.7%), followed by iPads (3.9%) and Macs (0.2%). The most popular device models are the Pro Max variants of iPhone 13/14/15/16.

Device slots. Device-slot utilization reflects the scale, availability, and signing capacity of developer-account pools. For 2,686 developer accounts with known credentials, 2,325 (86.6%) exhausted all 100 iPhone slots in the most recent year, whereas only 196 (7.3%) exhausted all iPad slots. On average, each account consumed 91 iPhone slots and 23 iPad slots.

Analyzing 2,405,833 ad-hoc provisioning profiles, we identified 10,843 distinct developer accounts. Each account registered 177.7 unique devices on average (median 20), and 7,088 accounts (65.4%) registered fewer than 100 devices. Meanwhile, 1,545 accounts (14.2%) remained active for at least one year. The 93 longest-lived accounts remained active for over 1,000 days, registering 710.0 devices on average. We also observed 295 accounts that registered more than 1,000 iPhone devices within one to two years, suggesting that the 100-device limit may sometimes be lifted or bypassed, though the underlying mechanism is unclear.

Activity. Fig. 7b shows persistently growing end-user demand for signing services (measured by request counts across all 258 platforms). Two major upstream suppliers, *SPEED* and *SUPERB*, account for 71.4% of all signing requests.

TABLE X: Device UDID occurrence counts and unique-device counts in ad-hoc provisioning profiles, with average, central 90%, and maximum lifespans.

		Lifespan		
#Occur.	**#Device UDIDs**	**Avg.**	**Central 90%**	**Max**
1	1,220,025 (71.3%)	291 d	32–365 d	365 d
2	274,217 (16.0%)	366 d	63–669 d	730 d
$\geq$3	216,068 (12.6%)	353 d	30–705 d	1455 d

(2b) Persistence of Abuse

Validity period. Among 2,405,833 ad-hoc provisioning profiles, 1,947,108 (80.9%) matched signing-request logs with plan annotations (Table III). Of these, 1,221,470 (62.7%) were zero-day warranty plans with an average lifespan of 280 days before expiration or revocation; 640,130 (32.9%) were 40-day plans (278 days); and 85,508 (4.4%) were 180- or 330-day plans (308 days). 37.8% profiles were revoked across all plans.

Certificate replacement. The gap between the number of provisioning profiles (2,405,833) and unique device UDIDs (1,710,310) indicates that many devices obtained multiple signing materials, presumably to restore access after certificate expiration or revocation.

Table X shows that 1,220,025 UDIDs (71.3%) appeared in only one profile, with an average lifespan of 291 days (central 90%: 32–365 days). Another 274,217 UDIDs (16.0%) appeared in two profiles (average 366 days, range 63–669 days), and 216,068 UDIDs (12.6%) appeared in three or more profiles (average 353 days, maximum 1,455 days). Thus, 28.7% of users repeatedly obtained replacement materials, enabling prolonged access to underground apps.

(2c) Account Source

Ad-hoc abuse scales by continuously acquiring new or reusable developer accounts. We examine account sources via email domains, geographic regions, and observed prices.

Email-address domains. Among 2,686 accounts, 1,134 (42.2%) use custom or private domains (*e.g.*, `**time.com` (314 accounts with the pattern), `5**.cn` (313), `t***m.cn` (117)), suggesting bulk registration with possibly leaked or fake identities. Accounts relying on public email services (*e.g.*, Gmail, Outlook) also often exhibit bulk patterns, such as random strings or names followed by numeric suffixes.

Countries and regions. Among 10,843 developer accounts extracted from provisioning profiles, the most common regions of the legal identities are China (4,793; 44.2%), the United States (1,451; 13.4%), Turkey (706; 6.5%), followed by Vietnam, India, Russia, Pakistan, and the United Kingdom.

Cost of accounts. Apple does not consistently enforce regional alignment between billing and identity verification [33]. This matters because Apple Developer Program fees differ substantially by region. Using the United States annual fee ($99 US Dollars) as baseline, we collected regional prices (using April 2026 exchange rates): lowest were Turkey

($14.4), Egypt ($29.5), Nigeria ($32.4), Pakistan ($56.9); medium regions included India ($92.3), United States, China ($100.4), United Kingdom ($106.7), Finland ($116.0).

3) Resilience Modeling: We model observed persistence using a renewal framework. For an app or device q, let its signing materials be ordered by issue time. Each material m is issued at s_m and remains valid until $\rho_m = \min(e_m, r_m)$, where e_m denotes its expiration time and r_m revocation (with $r_m = \infty$ if no revocation is observed). The total coverage time is thus $C_q = \mu(\bigcup_m [s_m, \rho_m])$, where μ denotes interval length. The extension factor $E_q = C_q / \max_m(\rho_m - s_m)$ quantifies the availability gained through replacement relative to the longest-lived single material. The recovery lag $\tau_m = \max(0, s_{m+1} - \rho_m)$ measures downtime before the next material becomes available. Together, E_q and τ_m capture the persistence gain and recovery speed, respectively.

VI. Mitigation

Implications for Apple. The ecosystem's vulnerability lies not in individual revoked certificates, but in their replacement paths. Our dataset exposes these paths through recurring links among developer accounts, certificates, provisioning profiles, device UDIDs, Bundle IDs, and entitlements. Apple should therefore track signing lineages rather than individual certificates. For ad hoc distribution, multiple UDIDs from a revoked developer account reappearing under the same successor indicate replacement of device-slot capacity. For enterprise distribution, recurrence of the same Bundle ID across successor developer accounts, along with rapid certificate churn or app diversity inconsistent with internal use, signals continued abuse. Revalidation should target linked account clusters to constrain signing-capacity replenishment.

Feasibility analysis shows these paths are detectable promptly. We define a revoked-to-successor developer account pair when two distinct devices transition from the revoked developer account to the same successor. Applying this rule to 150,061 post-revocation transitions, we identify 19,517 pairs; 84.0% reach the threshold within seven days of the first transition. These pairs account for 55,296 subsequent transitions (36.9% of all observed transitions), indicating successor developer accounts are identifiable while reuse is ongoing.

Implications for Law Enforcement. While Apple identifies replacement lineages via provisioning records, law-enforcement investigations often start with seized frontends. These frontends are entry points to upstream networks: shared source code, API endpoints, certificates, promoter accounts, and payment identifiers link independent storefronts to common intermediaries and account-pool operators. Reconstructing these dependencies reveals clusters controlling signing capacity, allowing disruption of one upstream cluster to affect multiple downstream services. Since account acquisition, infrastructure, payments, and operations may span jurisdictions, cross-border coordination is needed.

VII. Discussion

Novel Findings. Our central finding is that the *iASPD* ecosystem is organized around signing capacity as a continuously reallocated resource. Our operator-side data show how Apple's distribution constraints are converted into tradable units: ad hoc capacity is sold by device slot, enterprise capacity by app. Intermediaries route demand through reseller chains, and upstream pools replenish capacity after revocation. This allocation decouples service delivery from upstream control, explaining the ecosystem's layered structure and resilience: mechanisms that prior external studies could not observe.

Generalizability beyond China. Although our dataset is drawn from Chinese-facing infrastructure, the supply-chain structure we uncover — layered reselling, monetization of signing capacity, and certificate replacement — is not region-specific. Its constraints derive from Apple's globally uniform developer programs. The developer accounts and enterprise certificates in our records span multiple countries, and prior work documents comparable off-store abuse elsewhere [4], [5], [13], [34]. This evidence indicates that the actors and demand sustaining the ecosystem are not confined to China.

Limitations. Our seized dataset, obtained through law-enforcement collaboration, may not capture every operational variant of the *iASPD* ecosystem. Provider conversations cover only reachable sellers and serve as supplementary evidence. Records end at seizure, precluding observation of subsequent migration. App characterization using metadata and VirusTotal may introduce classification error. These limitations affect coverage and labeling, but do not undermine the central finding: the market's resilience stems from reusable upstream resources and coordinated replacement workflows that public-facing measurements cannot observe.

VIII. Related Work

We organize the related literature into four areas: certificate abuse for off-store distribution, certificate resale supply chains, revocation evasion, and underground abuse economies.

Developer Certificate Abuse for Off-Store Distribution. Prior work shows that Apple's developer and enterprise certificates, intended for testing or internal deployment, enable off-App-Store execution. Wei *et al.* identified enterprise provisioning as an attack vector on non-jailbroken devices [35]. Zheng *et al.* studied *Enpublic Apps* distributed with enterprise or developer certificates, often using private APIs or insecure communication [4]. Other studies document enterprise-app gatekeeper bypasses, WireLurker-style provisioning abuse, YiSpecter-style enterprise-signed malware, and third-party distribution [5], [13], [34], [36]–[38]. These works establish that Apple-recognized signing materials can be repurposed for distribution, but they focus on characterizing visible apps, campaigns, or channels, not the underlying provisioning and

routing infrastructure. Our work instead examines the operational ecosystem that supplies and allocates signing capacity.

Certificate Resale and Supply Chain. Liu *et al.* provide the closest prior view of certificate resale, showing how developer accounts are monetized through an ad hoc resale chain spanning signing websites, resellers, and upstream suppliers [6]. Their analysis, however, is reconstructed from user-facing sites, signing tools, and app categorization. It therefore infers actor layers but cannot confirm intermediary roles, trace request routing, observe cross-layer allocation and monetization of signing capacity, or examine coordinated recovery after revocation; enterprise signing is also beyond its scope. In contrast, our seized databases cover both ad hoc and enterprise signing, directly identify promoters, frontends, intermediaries, and upstream suppliers, and expose the routing and replacement workflows that connect them.

Revocation Evasion. Revocation is a key platform control: iOS validates certificates via OCSP, and revoked certificates render apps non-launchable [2], [39]. Prior work documents abusive operators' countermeasures: re-signing, renaming bundle metadata, changing device time, procuring additional certificates, or purchasing new accounts [4], [14], [34]. Public communities also describe DNS-based "anti-revoke" practices to block certificate-status validation [27], [28]. While these studies identify evasion techniques, they do not quantify how the underground market systematically replaces revoked resources upstream.

Underground Abuse Economies. A broader line of research studies cybercrime as a specialized underground economy. Levchenko *et al.* analyze the spam value chain, revealing its dependence on naming, hosting, payment, and fulfillment components [40]. Caballero *et al.* measure pay-per-install malware distribution, where clients buy installs from PPI providers and affiliates supply compromised hosts [41]; Thomas *et al.* further study commercial PPI and unwanted-software distribution at scale [42]. McCoy *et al.* use leaked pharmaceutical-affiliate records to analyze advertisers, transactions, revenues, and market structure [43]. Thomas *et al.* model criminal markets as profit and support centers that sell specialized services and resources [44]. Van Wegberg *et al.* measure commoditized cybercrime offerings on anonymous markets and develop a value-chain view of cybercrime components [45]. Bhalerao *et al.* reconstruct cybercrime supply chains from underground forum interactions [46]. Unlike our work, these studies focus on other abuse domains (*e.g.*, spam, PPI, pharmaceuticals).

IX. Conclusion

In this work, we present an inside measurement of the underground *iASPD* ecosystem using seized databases from 258 operations (177 domains), supplemented by 60 e-commerce merchants and 10 provider conversations. Our findings show that unauthorized iOS distribution is not isolated abuses, but a layered ecosystem: developers and users generate demand; promoters attract customers; frontends provide signing/hosting; intermediaries route signing capacity; and upstream suppliers maintain account pools. Apple's constraints shape monetization: ad hoc signing is sold per device slot, in-house per app. Revocation is absorbed via certificate replacement and rerouting, making it a recoverable event. These findings reveal the market's structure and identify intervention points for platform defense and law enforcement.

X. Ethical Considerations

This study follows the principles of the Menlo Report [47]. It involves sensitive operational data from an underground iOS signing ecosystem, including seized backend databases, e-commerce observations, provider conversations, and app analysis. We designed the study to minimize privacy risk, avoid operational harm, and prevent reproduction of abusive services; the study was reviewed by our institution's ethics committee and conducted with authorization from the collaborating law-enforcement agencies.

Data Provenance. Seized materials were obtained through law enforcement collaboration under institutional authorization. Our analysis focuses on ecosystem measurement: app distribution, signing flows, reseller relationships, payment patterns, and revocation recovery. We do not assess individual criminal liability and avoid identifying specific users, merchants, or providers unless information is already public and necessary for defensive interpretation.

Sensitive Data Handling. Raw identifiers (phone numbers, emails, UDIDs, payment records, *etc.*) are access-controlled and de-identified; sensitive fields are pseudonymized from figures and examples. Results are reported in aggregate. We disclose no real service names, domains, payment accounts, certificates, provisioning profiles, private keys, or credentials.

Non-use of Operational Materials. We do not use recovered credentials, certificates, or provisioning profiles to access Apple services, register devices, sign or install apps, or test services. Analysis is performed solely on archived metadata and database relationships.

Online Conversation Protocols. Because the relevant operational details were not publicly observable, we posed as potential customers or brokers in online conversations with service providers. These conversations did not involve end users and were limited to pricing plans, operational scale, resource availability, and claimed account sources. We made no purchases or payments, submitted no apps, requested no malicious actions, and did not request or use credentials, certificates, provisioning profiles, or other operational materials.

Disclosure and Artifact Release. We responsibly disclosed relevant findings and sanitized datasets to Apple. Public release is limited to sanitized aggregate statistics organized under the integrated schemas. We withhold the seized databases, source code, raw conversations, credentials, certificates, and other signing materials because they could identify participants or enable reconstruction of abusive services. This restriction limits full reproducibility but is necessary to prevent the released artifacts from facilitating further abuse.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (No.62272410).

REFERENCES

[1] Apple Developer Documentation, "Distributing Your App to Registered Devices," https://developer.apple.com/documentation/xcode/distributing-your-app-to-registered-devices.

[2] Apple Support, "Distribute Proprietary In-House Apps to Apple Devices," https://support.apple.com/guide/deployment/distribute-proprietary-in-house-apps-depce7cefc4d/web.

[3] Apple Newsroom, "The App Store prevented more than $9 billion in fraudulent transactions over the last five years," https://www.apple.com/newsroom/2025/05/the-app-store-prevented-more-than-9-billion-usd-in-fraudulent-transactions/.

[4] M. Zheng, H. Xue, Y. Zhang, T. Wei, and J. C. Lui, "Enpublic apps: Security threats using ios enterprise and developer certificates," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 463–474.

[5] Y. Guo, D. Wang, L. Wang, Y. Fang, C. Wang, M. Yang, T. Liu, and H. Wang, "Beyond App Markets: Demystifying Underground Mobile App Distribution Via Telegram," in *Abstracts of the 2025 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2025, pp. 100–102.

[6] Y. Liu, Y. Zhang, B. Liu, and H. Duan, "Cracks in the Walled Garden: Dissecting the Gray-Market of Unauthorized iOS App Distribution via Ad Hoc Sideloading," in *35th USENIX Security Symposium (USENIX Security 26)*. Baltimore, MD: USENIX Association, Aug. 2026.

[7] Y. Lee, X. Wang, X. Liao, and X. Wang, "Understanding illicit UI in iOS apps through hidden UI analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2390–2402, 2019.

[8] Y. Zhao, L. Yu, Y. Sun, Q. Liu, and B. Luo, "No source code? no problem! demystifying and detecting mask apps in ios," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, 2024, pp. 358–369.

[9] H. Lin, Y. Hu, H. Xu, Y. Lu, M. Ren, S. Hao, C. Yue, Z. Li, F. Zhang, and Y. Jiang, "CHAMELEOSCAN: Demystifying and Detecting iOS Chameleon Apps via LLM-Powered UI Exploration," in *Proceedings of the 33rd Network and Distributed System Security Symposium (NDSS 2026)*, 2026.

[10] Sophos, "Beware bogus Betas – cryptocoin scammers abuse Apple's TestFlight system," https://www.sophos.com/en-us/blog/beware-bogus-betas-cryptocoin-scammers-abuse-apples-testflight-system.

[11] Sublime Security, "Fake Meta Ads Manager in App Store and TestFlight used to phish Meta ad accounts," https://sublime.security/blog/fake-meta-ads-manager-in-app-store-and-testflight-used-to-phish-meta-ad-accounts/.

[12] Sophos, "Fake Android and iOS apps disguise as trading and cryptocurrency apps," https://www.sophos.com/en-us/blog/fake-android-and-ios-apps-disguise-as-trading-and-cryptocurrency-apps.

[13] A. Bashan and O. Bobrov, "Enterprise Apps: Bypassing the iOS Gatekeeper," Black Hat Asia 2016. https://blackhat.com/docs/asia-16/materials/asia-16-Bashan-Enterprise-Apps-Bypassing-The-iOS-Gatekeeper-wp.pdf, 2016.

[14] Apple White Paper, "Building a Trusted Ecosystem for Millions of Apps: A Threat Analysis of Sideloading," https://www.apple.com/privacy/docs/Building_a_Trusted_Ecosystem_for_Millions_of_Apps.pdf.

[15] Q. Hu, S. Wu, W. Sun, Z. Tang, C. Chen, Z. Ding, and X. Zhang, "Measurement of the usage of web clips in underground economy," *arXiv preprint arXiv:2209.03319*, 2022.

[16] Apple Developer, "Distribute on the App Store," https://developer.apple.com/app-store/, 2026.

[17] ——, "TestFlight Overview," https://developer.apple.com/help/app-store-connect/test-a-beta-version/testflight-overview/, 2026.

[18] ——, "App Review," https://developer.apple.com/distribute/app-review/, 2026.

[19] Apple Developer Documentation, "Devices Overview," https://developer.apple.com/help/account/devices/devices-overview/, 2026.

[20] ——, "Program Enrollment – Apple Developer Account Help," https://developer.apple.com/help/account/membership/program-enrollment/, 2026.

[21] ——, "Device Registration Updates," https://developer.apple.com/help/account/reference/device-registration-updates/, 2026.

[22] ——, "Enabling Developer Mode on a Device," https://developer.apple.com/documentation/xcode/enabling-developer-mode-on-a-device, 2026.

[23] Apple Developer, "Apple Developer Enterprise Program," https://developer.apple.com/programs/enterprise/, 2026.

[24] ——, "Apple Developer Enterprise Program License Agreement," https://developer.apple.com/support/downloads/terms/apple-developer-enterprise-program/Apple-Developer-Enterprise-Program-License-Agreement-English.pdf, 2026.

[25] Apple Support, "Web Clips Device Management Payload Settings for Apple Devices," https://support.apple.com/guide/deployment/web-clips-payload-settings-depbc7c7808/web, 2026.

[26] Apple Platform Security, "App Code Signing Process in iOS, iPadOS, tvOS, visionOS, and watchOS," https://support.apple.com/guide/security/app-code-signing-process-sec7c917bf14/web, 2026.

[27] R. Chouhan, "iOS Anti Revoke Guide: NextDNS Block URLs List Setup," https://blogs.rohitchouhan.com/hacks/ios-anti-revoke-esign-feather-nextdns-block-urls-guide/.

[28] Builds.io, "The Free Certificate Method for iOS Sideloading: A Complete Guide," https://builds.io/blog/technologies/ios-technologies/the-free-certificate-method-for-ios-sideloading/, 2025.

[29] S. Puzan and D. Kalinin, "SparkKitty, SparkCat's Little Brother: A New Trojan Spy Found in the App Store and Google Play," https://securelist.com/sparkkitty-ios-android-malware/116793/, 2025.

[30] C. Xiao, "iOS Trojan "TinyV" Attacks Jailbroken Devices," https://unit42.paloaltonetworks.com/ios-trojan-tinyv-attacks-jailbroken-devices/, 2015, palo Alto Networks Unit 42. Published Dec. 15, 2015.

[31] AppleP12.com, "AppleP12.com Telegram Channel," https://t.me/s/AppleP12, 2026, public Telegram archive containing iOS certificate package and certificate-status posts. Accessed May 2026.

[32] NovaDev404, "NovaCerts: The Best Free iOS Certificates Repo," https://github.com/NovaDev404/NovaCerts, 2026, gitHub repository aggregating free iOS signing certificates; README states that sources include Loyahdev, NezusHub/NabzClan, AppleJR, Khoindvn, and Eojoo, and that many certificates originate from AppleP12 Telegram.

[33] Reddit user report, "How I Enrolled in the Apple Developer Program After "Your Enrollment Could Not Be Completed"," https://www.reddit.com/r/iOSProgramming/comments/kgyuex/how_i_enrolled_in_the_apple_developer_program/.

[34] A. Khanlari and A. Rahmati, "Taking a Bite Out of the Forbidden Fruit: Characterizing Third-Party Iranian iOS App Stores," *arXiv preprint arXiv:2604.26343*, 2026.

[35] T. Wei, M. Zheng, H. Xue, and D. Song, "Apple without a shell–iOS under targeted attack," in *Virus Bulletin Conference*, 2014.

[36] Palo Alto Networks Unit 42, "WireLurker: A New Era in iOS and OS X Malware," https://www.paloaltonetworks.com/resources/research/unit42-wirelurker-a-new-era-in-ios-and-os-x-malware.html, 2014.

[37] C. Xiao, "YiSpecter: First iOS Malware That Attacks Non-jailbroken Apple iOS Devices by Abusing Private APIs," https://unit42.paloaltonetworks.com/yispecter-first-ios-malware-attacks-non-jailbroken-ios-devices-by-abusing-private-apis/, 2015.

[38] Lookout, "Managing App Sideloading Threats on iOS," https://info.lookout.com/rs/051-ESQ-475/images/Managing-iOS-App-Sideloading-USv2.1.pdf, 2015.

[39] Apple Developer Support, "Certificates," https://developer.apple.com/support/certificates/, 2025.

[40] K. Levchenko, A. Pitsillidis, N. Chachra, B. Enright, M. Félegyházi, C. Grier, T. Halvorson, C. Kanich, C. Kreibich, H. Liu *et al.*, "Click trajectories: End-to-end analysis of the spam value chain," in *2011 ieee symposium on security and privacy*. IEEE, 2011, pp. 431–446.

[41] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring Pay-per-Install: The commoditization of malware distribution," in *20th USENIX Security Symposium (USENIX Security 11)*, 2011.

[42] K. Thomas, J. A. E. Crespo, R. Rasti, J.-M. Picod, C. Phillips, M.-A. Decoste, C. Sharp, F. Tirelo, A. Tofigh, M.-A. Courteau *et al.*, "Investigating Commercial Pay-Per-Install and the Distribution of Unwanted Software," in *25th USENIX Security Symposium*, 2016, pp. 721–739.

[43] D. McCoy, A. Pitsillidis, J. Grant, N. Weaver, C. Kreibich, B. Krebs, G. Voelker, S. Savage, and K. Levchenko, "Pharmaleaks: Understanding the business of online pharmaceutical affiliate programs," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 1–16.

[44] K. Huang, D. GrierD, T. J. Holt, C. Kruegel, D. McCoy, S. Savage, and G. Vigna, "Framing dependencies introduced by underground commoditization," in *Workshop on the Economics of Information Security*, 2015.
[45] R. Van Wegberg, S. Tajalizadehkhoob, K. Soska, U. Akyazi, C. H. Ganan, B. Klievink, N. Christin, and M. Van Eeten, "Plug and prey? measuring the commoditization of cybercrime via online anonymous markets," in *27th USENIX security symposium*, 2018, pp. 1009–1026.
[46] R. Bhalerao, M. Aliapoulios, I. Shumailov, S. Afroz, and D. McCoy, "Mapping the underground: Supervised discovery of cybercrime supply chains," in *2019 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2019, pp. 1–16.
[47] D. Dittrich and E. Kenneally, "The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research," U.S. Department of Homeland Security, Tech. Rep., August 2012.

APPENDIX

A. Dataset Schema

Tables XI and XII define the field dictionary for the eight unified tables. Each row specifies the table, field name, semantic type, key role, and description. In the *Type* column, "?" denotes an optional field, "[]" a list-valued field, and `Time` fields are UTC timestamps. In the *Key* column, `PK`, `FK`, and `G` denote primary keys, foreign keys, and reusable grouping identifiers, respectively. "Redacted." indicates values withheld from public release. The full dataset and additional documentation are available at https://doi.org/10.5281/zenodo.22216862.

TABLE XI: Dataset field dictionary (Part I): developer accounts and certificates.

Tbl.	Field	Type	Key	Description
developer_accounts	developer_account_id	ID	PK	Stable identifier for a developer-account credential record within one platform.
	platform_id	ID	G	Identifier of the platform on which the account was observed.
	platform_name	Label	-	Redacted. Human-readable name of the platform.
	developer_name	Label	-	Redacted. Name associated with the developer identity.
	account_name	Label?	-	Redacted. Account name when distinct from the developer name.
	email	Label?	-	Redacted. Primary email address associated with the account.
	alternate_email	Label?	-	Redacted. Secondary email address when distinct from the primary address.
	team_id	ID?	G	Stable identifier for the Apple developer team associated with the account.
	used_iphone_slot_count	Count?	-	Number of used iPhone device-registration slots (0–100).
	used_ipad_slot_count	Count?	-	Number of used iPad device-registration slots (0–100).
	used_mac_slot_count	Count?	-	Number of used Mac device-registration slots (0–100).
	valid_until	Time	-	Time at which the account's recorded validity ends.
certificates	certificate_id	ID	PK	Stable identifier for a distinct certificate.
	certificate_name	Label?	-	Redacted. Certificate subject common name or, when unavailable, its recorded descriptive label.
	developer_account_id	ID?	FK	Developer account associated with the certificate, when uniquely identified.
	team_id	ID?	G	Stable identifier for the Apple developer team associated with the certificate.
	certificate_type	Enum?	-	Certificate class: `development` or `distribution`.
	issued_at	Time?	-	Start of the certificate validity period.
	expires_at	Time?	-	End of the certificate validity period.
	revocation_status	Enum?	-	Most recent observed status: `good`, `revoked`, `unknown`, or `error`.
	status_observed_at	Time?	-	Time at which `revocation_status` was observed.
	revoked_at	Time?	-	Recorded revocation time when `revocation_status` is `revoked`.

TABLE XII: Dataset field dictionary (Part II): provisioning profiles, signing requests, registered accounts, payment orders, pricing plans, and apps.

Tbl.	Field	Type	Key	Description
provisioning_profiles	provisioning_profile_id	ID	PK	Stable identifier for a distinct provisioning-profile artifact.
	logical_profile_id	ID?	G	Identifier shared by artifacts representing the same logical provisioning profile.
	profile_name	Label?	-	Redacted. Descriptive name of the provisioning profile.
	application_identifier_id	ID	G	Identifier shared by profiles with the same application identifier.
	application_identifier	Label	-	Redacted. Application identifier specified by the profile.
	primary_certificate_id	ID?	FK	Primary signing certificate embedded in the profile, when identified.
	app_id	ID?	FK	Application covered by the profile, when linked to a released app.
	platform_id	ID	G	Identifier of the platform from which the profile was observed.
	platform_name	Label	-	Redacted. Human-readable name of that platform.
	team_id	ID	G	Stable identifier for the Apple developer team specified by the profile.
	profile_type	Enum?	-	Profile class: `ad_hoc` or `in_house`.
	created_at	Time	-	Profile creation time.
	expires_at	Time	-	Profile expiration time.
	provisions_all_devices	Bool?	-	Whether the profile permits installation on all devices.
	registered_device_count	Count	-	Number of distinct devices listed in the profile.
	registered_device_ids	ID[]	G	Stable identifiers of the devices listed in the profile.
signing_requests	signing_request_id	ID	PK	Stable identifier for a deduplicated signing request.
	platform_id	ID	G	Identifier of the intermediary platform that recorded the request.
	platform_name	Label	-	Redacted. Human-readable name of that intermediary platform.
	requester_account_id	ID?	FK	Registered account that submitted the request, when identified.
	registered_device_id	ID	G	Stable identifier of the device registered for the request.
	warranty_days	Dur.	-	Warranty or replacement period, in days.
	signing_developer_account_id	ID?	FK	Developer account used to sign the app, when identified.
	signing_certificate_id	ID	FK	Certificate used to sign the app.
	issued_profile_id	ID	FK	Provisioning profile issued for the request.
	submitted_app_id	ID?	FK	Application submitted for signing, when linked to a released app.
	request_observed_at	Time	-	Earliest recorded time associated with the request.
registered_accounts	registered_account_id	ID	PK	Stable identifier for a platform account record.
	platform_id	ID	G	Identifier of the platform on which the account was registered.
	platform_name	Label	-	Redacted. Human-readable name of that platform.
	account_context	Enum	-	Registration context: service frontend, intermediary, or registered reseller.
	username	Label?	-	Redacted. Username associated with the account.
	email	Label?	-	Redacted. Email address associated with the account.
	registered_at	Time?	-	Recorded registration time for the account.
	last_login_at	Time?	-	Most recent recorded login time for the account.
payment_orders	payment_order_id	ID	PK	Stable identifier for a payment-order record.
	platform_id	ID	G	Identifier of the platform that recorded the order.
	platform_name	Label	-	Redacted. Human-readable name of that platform.
	order_reference	Label	-	Redacted. Order or payment reference associated with the record.
	purchaser_account_id	ID?	FK	Registered account that placed the order, when identified.
	order_amount	Amount	-	Recorded monetary amount of the order.
	currency	Curr.	-	Currency of the order amount; `CNY` in this dataset.
	ordered_at	Time	-	Time at which the order was created.
	paid_at	Time	-	Time at which payment was completed.
pricing_plans	pricing_plan_id	ID	PK	Stable identifier for a pricing-plan observation.
	platform_id	ID	G	Identifier of the platform offering the plan.
	platform_name	Label	-	Redacted. Human-readable name of that platform.
	provider_code	Code?	-	Provider code used in the per-app comparison (A–E).
	provider_type	Enum	-	Role of the seller or provider.
	plan_name	Label	-	Redacted. Advertised name of the plan.
	plan_type	Enum	-	Category of the offered plan.
	pricing_model	Enum	-	Pricing basis: `per_device` or `per_app`.
	device_type	Enum?	-	Eligible device type for a per-device plan.
	warranty_days	Dur.?	-	Free-replacement period for a per-device plan, in days.
	price_unit	Enum	-	Unit to which the price applies: device slot, app-month, download, or day.
	list_price_amount	Amount	-	Listed price, or the lower endpoint when the price is a range.
	list_price_upper_amount	Amount?	-	Upper endpoint of a listed price range.
	currency	Curr.	-	Currency of the listed price: `CNY` or `USDT`.
apps	app_id	ID	PK	Stable identifier for a distinct native iOS app.
	base_app_id	ID?	G	Identifier shared by variants of the same base app.
	app_name	Label?	-	Redacted. Name of the app.
	bundle_id	Label?	-	Redacted. Bundle identifier of the app.
	platform_ids	ID[]	G	Identifiers of all platforms on which the app was observed.
	distribution_methods	Enum[]	-	Observed distribution methods: `unsigned`, `ad_hoc_signed`, or `enterprise_signed`.
	functional_category	Category	-	Primary functional category assigned to the app.
	secondary_functional_categories	Category[]?	-	Additional functional categories assigned to the app.
	abusive_category	Category?	-	Dominant advertised abusive behavior associated with the app.
	signing_tool_observed	Bool	-	Whether the app metadata indicates signing-tool functionality.
	app_store_name	Label?	-	Redacted. Name of the matched App Store listing.
	app_store_category	Category?	-	Primary category of the matched App Store listing.
	app_store_paid	Bool?	-	Whether the matched App Store listing is paid.
	app_store_price	Amount?	-	Price of the matched App Store listing.
	app_store_currency	Curr.?	-	Currency of the matched App Store price.
	first_seen_at	Time	-	Earliest recorded observation of the app.
	last_seen_at	Time	-	Latest recorded observation of the app.

APPENDIX A
ARTIFACT APPENDIX

This appendix describes the dataset accompanying "Beyond the App Store: Uncovering the Underground iOS Signing Supply Chain via Operational Data."

A. Description & Requirements

The artifact provides a sanitized research dataset for the iOS Abusive Signing-Provisioning-Distribution (iASPD) ecosystem. It comprises eight unified CSV tables covering apps, signing activity, developer accounts, platform accounts, payments, and pricing plans. The tables follow the unified data model described in Section IV-C and Appendix B of the paper. They document the entities and relationships examined in our study of app distribution, intermediary resale, and upstream signing resources.

The release includes `README.md`, `SCHEMA.md`, and a Markdown guide for each table. The schema specifies field definitions, construction rules, value representations, release totals, and relationships. The table guides describe the corresponding populations and provide examples of how to interpret individual records.

1) How to access: The dataset and accompanying documentation are provided as a ZIP archive. The Zenodo record is https://doi.org/10.5281/zenodo.22216862.

2) Hardware dependencies: No specialized hardware is required. The extracted release occupies approximately 0.8 GB of disk space.

3) Software dependencies: A ZIP extraction utility and a CSV reader with UTF-8 support are sufficient for accessing the data. The Markdown documentation can be read in a text editor.

4) Benchmarks: The eight released tables contain the following data.

Signing resources. `developer_accounts.csv` contains 2,686 platform-scoped developer-account records, including device-slot usage and recorded validity boundaries. `certificates.csv` contains metadata for 19,285 signing-certificate identities, with account and team links, validity intervals, and the most recent recorded revocation-status observations. The status fields describe the evidence at the recorded observation time. `provisioning_profiles.csv` contains 2,407,269 profile-artifact records with provisioning modes, creation and expiration times, registered-device identifiers, and certificate links. The profile table distinguishes exact artifact identities from logical profile identities: several artifacts can share one logical identity. The release contains 2,406,463 distinct populated logical profile identifiers.

Signing requests. `signing_requests.csv` contains 395,637 shared platform-side signing requests. Each row represents a deduplicated request at one intermediary and records its device identifier, warranty duration, signing certificate, issued profile, and observation time. Observations at different intermediaries remain separate rows. Requester, developer-account, and app links are recorded where the corresponding relationships are resolved.

Accounts and pricing. Account records are stored in `registered_accounts.csv`. Its 39,242 rows comprise 31,955 base-account contexts at service frontends or intermediaries and 7,287 registered-reseller contexts. The payment table, `payment_orders.csv`, contains 206,536 successful orders with recorded amounts, order and payment times, and purchaser links. Pricing observations are stored in `pricing_plans.csv`, which contains 895 plans covering per-device offers and per-app distribution fees. Its fields specify the provider role, plan type, applicable device and warranty period, price unit, and currency. Payment amounts are recorded in CNY; plan prices use CNY or USDT according to the fee type.

Apps. `apps.csv` contains 22,259 native iOS app entities. It records observed distribution methods, functional and abusive categories, App Store matching attributes, and observation intervals. Variants remain separate app rows and are associated with base-app families through a grouping identifier. Platform and distribution-method memberships are represented as lists, since an app may occur on multiple platforms or under multiple distribution methods. App Store attributes describe the matched base-app family, while functional and abusive categories describe the observed app metadata.

B. Artifact Installation & Configuration

Download and extract the archive, then open `README.md` for the file index. Consult `SCHEMA.md` for column definitions and the corresponding table guide for population and interpretation details. The CSV tables can be read directly; no artifact-specific installation or configuration is required.

When importing a table, select UTF-8 encoding and comma separation, with the first row interpreted as column names. Timestamps are represented in UTC, Boolean values as `true` or `false`, and list-valued fields as JSON arrays within individual CSV cells. Empty cells represent unavailable or unresolved values and are distinct from numeric zero, `false`, and an empty array. Monetary values should be interpreted together with their currency and, for pricing plans, their price unit.

The tables can be joined through the relationships documented in `SCHEMA.md`. For example, signing requests link to their issued profiles and signing certificates, and payment orders link to purchaser account records. These joins use stable identifier columns rather than masked display labels. Grouping identifiers for platforms, developer teams, devices, and base-app families can be used to aggregate records within the corresponding identifier domain. The schema distinguishes these grouping identifiers from primary and foreign keys, and indicates which relationships are optional.