PermiOSAn: Mapping and Measuring iOS and Android Permissions Across Space and Time

Magdalena Steinböck
TU Wien
magdalena.steinboeck@seclab.wien

Jakob Bleier
TU Wien
jakob.bleier@seclab.wien

Florian Draschbacher
TU Graz and A-SIT Austria
florian.draschbacher@tugraz.at

Christine Utz
Radboud University
christine.utz@ru.nl

Tobias Urban
Institute for Internet Security
urban@internet-sicherheit.de

Martina Lindorfer
TU Wien
martina@seclab.wien

Abstract

Mobile operating systems like Android and iOS have established permission systems as a security and privacy measure to protect access to sensitive resources. While these permission systems are conceptually similar, their implementations differ, making comparisons across platforms non-trivial. For example, it is unclear if results on the privacy impact of an app on iOS hold true for the same app on Android (e.g., the app might use a different set of permissions in both ecosystems despite identical feature set) or vice versa. To conduct holistic comparisons of apps across platforms, and the app ecosystems as a whole, it is vital to understand platform-specific differences in permission systems, whether similar permissions provide equivalent access to resources, and if there are discrepancies in permission-protected functionality.

In this paper, we introduce PermiOSAn, the first taxonomy of permissions accessible to third-party apps on both Android (n = 351) and iOS (n = 160). We derive 22 *Cross-Platform Permission Categories* (XPPCs), mapping 257 (73%) Android permissions to 50 (31%) iOS permissions. We apply our taxonomy to 2,964 pairs of cross-platform apps available on both Android and iOS between 2023 and 2025. Our results show that iOS apps generally request more XPPCs, while Android apps exhibit faster growth in the number of XPPCs requested and in the adoption rate across almost all categories. Our findings provide the foundation for future cross-platform studies and offer new insights to support privacy-friendly cross-platform development and more transparent choices for users.

1 Introduction

Smartphones have evolved from static communication devices into personalized computing platforms that users rely on for a wide range of activities, from social networking and entertainment to healthcare and financial services. Given the sensitive nature of users' personal data, modern mobile platforms like Google's Android and Apple's iOS have introduced permission systems that govern how third-party applications (apps) can access system resources and user data. In this paper, we perform the first in-depth comparison of how Android and iOS use permissions to restrict access to resources, investigate how these permission systems have evolved, and measure how apps have used these permissions across a cross-platform app dataset. Our results shed light on different notions of privacy protection and control, platform responsibility, and the abstraction of sensitive resources between Android and iOS.

To mediate access to resources, such as user data (e.g., contacts), system information (e.g., device identifiers), device sensors (e.g., location), hardware components (e.g., camera), or communication capabilities (e.g, Bluetooth connectivity), Android and iOS rely on their permission systems. While low-risk resources may have permissions that are granted automatically by the OS or not protected by a permission at all, access to more sensitive resources is typically safeguarded by user-controlled permissions, i.e., the user has to actively grant an app access during runtime [29, 48]. In addition, both OSs allow users to review and revoke granted permissions. Thus, permissions are a privacy feature that gives users visibility into and partial control over who can access their personal data.

Both Google and Apple publicly portray their platforms as offering superior privacy and security protections compared to the other [33, 74], based on architectural and governance choices as evidence of stronger user control, particularly in the design and evolution of their permission models. Yet, there is limited academic work to confirm or refute those claims. At the app level, the majority of related work has focused on privacy issues on Android [7, 93, 94, 96, 102], citing its openness and tool support. On the system level, Android's permission model has been studied in-depth since its inception [44, 114], but its counterpart on iOS has only recently been investigated, and only based on individual iOS-only permissions [92, 106]. The only attempt to compare both permission systems concerned apps' access to security-sensitive APIs on Android 4.0 and iOS 5.0 [76]. Later cross-platform studies have limited their analysis to selected permissions existing in both OSs [85] or focused on non-technical privacy measures, such as developer [75] and user awareness [36, 101], and privacy disclosures through app store labels [83, 104] and in-app consent dialogs [84, 95].

To close the gap between conceptual and implementation-specific differences in the platforms' permission models, we introduce fine-grained *Cross-Platform Permission Categories* (XPPCs) that map permissions across the latest Android and iOS versions. This taxonomy can serve as the foundation for (1) evaluating the impact of platform-level governance of sensitive resources, (2) providing developer guidance on how to develop secure and privacy-friendly apps, (3) comparing the behavior of apps across platforms, and (4) assessing the degree of agency provided to users in granting, denying, or revisiting resource access.

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–25

https://doi.org/XXXXXXX.XXXXXXX

In summary, we address the following research questions:

- **RQ1**: *Which permissions exist on Android and iOS, and can they be mapped across platforms?* We introduce a method to map 257 Android permissions to 50 iOS permissions that provide equivalent access to the same protected resources. Mapping all permissions available to third-party developers in Android 16 and iOS 26 Beta, we identify 22 XPPCs that constitute our extensible PermiOSAn taxonomy (Section 3).
- **RQ2**: *What are the differences in permission models between Android and iOS?* We identify permissions that currently have no counterpart on the other platform and use them to highlight feature disparity and platform-specific differences in how Android and iOS protect sensitive resources, the granularity of individual permissions, and platform policies (Section 3.5).
- **RQ3**: *Do the same apps request different (or more) permissions on one platform than on the other?* We apply our taxonomy to 2,964 cross-platform app pairs. Our results show notable differences in the requested permissions, suggesting that studies focusing on a single ecosystem may lack generalizability (Section 4).

2 The Android and iOS Permission Systems

To protect user data and system functionality, Android and iOS implement permission systems that guard sensitive data (e.g., user contact information or system state) and sensitive actions (e.g., access to hardware features or system functions) [18, 29, 48].

iOS differentiates between *protected resources*, which require user authorization, and *entitlements*, predefined key-value pairs that grant an app high-level capabilities or privileges (e.g., push notifications). The OS monitors access to protected resources and requires apps to provide a *rationale*, an explanatory string displayed in the permission dialog, in the `Info.plist` file. Entitlements are configured in a separate plist file (`.entitlements`) and embedded into the app's code signature. Some restricted entitlements require Apple approval before distribution. We use "permission" to encompass both permissions and entitlements, unless stated otherwise.

Android protects access to all restricted resources through a unified permission model based on user authorization. Apps must declare their required permissions, identified by unique names, in the `AndroidManifest.xml`, which provides the OS with the necessary information about the app. Unlike iOS, Android does not enforce rationales and only advises developers to explain to users why a permission is needed before spawning the consent dialog [64].

Both Android and iOS identify OS versions with (increasing) version numbers (e.g., Android 16). In 2025, Apple changed its iOS version numbering scheme to a year-based system, starting with iOS 26 for the 2025/26 release cycle [41]. This resulted in version numbering jumping from 18.6.1 to 26. On Android, developers must specify two values that determine their app's compatibility: the `targetSdkVersion` (the API level of the Android version the app is designed for) and the `minSdkVersion` (the oldest API level the app can run on). The API level ("technical level") is tied to the Android version ("user level") but identifies the Android platform and thus the available interfaces and resources. Unlike iOS, an Android app built for the latest OS version may execute on older versions and access deprecated APIs, as long as its `minSdkVersion` permits it.

2.1 Shared Concepts in Permission Handling

Both mobile OSs rely on permission mechanisms to regulate application access to sensitive resources. The first versions of Android used install-time permissions, where the user implicitly granted all permissions an app required during installation. In 2015, Android added runtime user prompts for particular permissions.

Runtime Permissions & Permission Dialogs. Nowadays, on both OSs, apps must request access to key resources through runtime user permission prompts. On Android, an app checks whether it has a runtime permission and can initiate a system dialog that asks the user to grant the permission ("permission dialog"). iOS monitors access to critical resources and shows a permission dialog to the user on first access. Permission dialogs allow users to control the app's resource usage. Most dialogs provide binary choices ("allow" or "deny"), yet certain permissions, such as location access, offer granular options (e.g., "allow only once" or "allow while the app is in use"). On Android, the system defines the text shown in the permission dialog. On iOS, the permission dialog includes developer-provided text explaining why access is required (rationale) as defined in the app's `Info.plist` file. Both systems provide centralized permission controls in the system settings, enabling users to update previously granted permissions.

Permission-granting Mechanisms. *Permission-granting mechanisms* specify the entities and processes that authorize an app's access to protected resources. Depending on the platform and permission type, permissions may be granted by the user, the system, the device or OS vendor, app store, or a combination thereof. User-granted permissions provide users with direct control over sensitive resources, whereas system- or vendor-granted permissions are typically reserved for privileged functionality and trusted apps. Store-mediated mechanisms add another layer of review by restricting the distribution of apps that request certain permissions. These different granting mechanisms also imply fundamentally different security and privacy threat models. For user-granted permissions, the primary risk is that users unknowingly or unintentionally authorize access to sensitive resources. By contrast, system-, vendor-, or store-granted permissions shift the trust decision from the user toward the OS, device vendor, or app marketplace, so the control of these permissions depends on the correctness of the platform's enforcement mechanisms and review processes rather than user consent. Thus, two permissions may provide similar resource access across OSs, but security and privacy implications can greatly differ due to different permission-granting mechanisms.

2.2 Differences in Permission Handling

Despite these commonalities, other aspects of the permission systems in Android and iOS differ considerably.

Protection Levels. On Android, each permission is assigned to a *protection level*. Permissions considered sensitive are mapped to the *dangerous* protection level, which requires user consent through permission dialogs on modern OS versions. Further levels exist, such as *normal*, which grants less powerful permissions at install time, and *privileged*, which restricts actions with far-reaching consequences to a subset of preinstalled apps. Google uses the terms *runtime permission* and *dangerous permission* interchangeably.

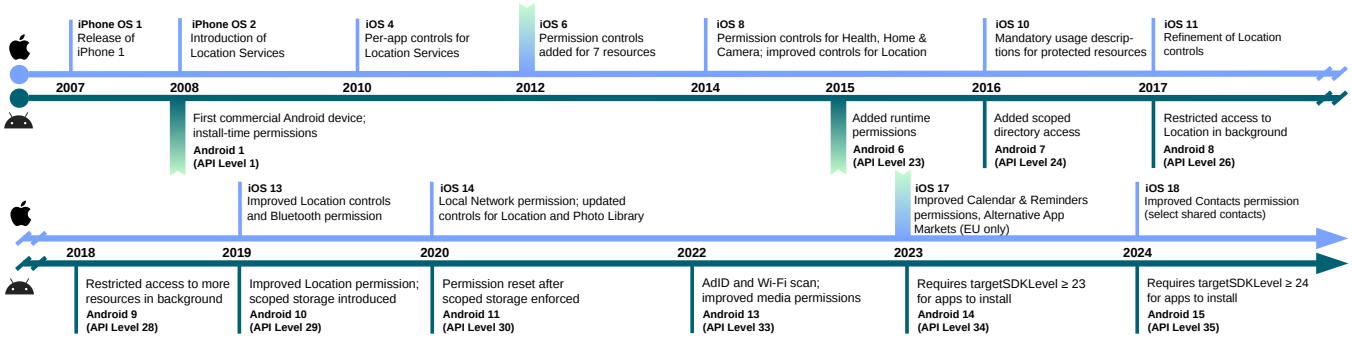

Figure 1: Milestones in the permission system changes in Android and iOS. Most notably, Android introduced install-time permissions at launch in 2008 (Android 1) and runtime permissions in 2015 (Android 5). iOS added runtime permissions for selected resources in 2012 (iOS 6) and, in 2023, allowed alternative app marketplaces in the EU (iOS 17).

Permission Groups. On Android, *permission groups* bundle related permissions and resource requests [55]. This means that granting one dangerous permission within a permission group leads to all permissions in the group being granted when requested by the app.

Restricted Permissions. On Android, access to certain sensitive data and APIs is further protected through *restricted permissions* [56], e.g., call log permissions. These permissions impose additional compliance requirements on developers and, for apps distributed via Google Play, require submission of a declaration form justifying their use, which is subject to Google's review [66].

Entitlements. All iOS apps need to include a valid code signature, which is applied by Apple for App Store distribution, the most common scenario. Embedded in this signature is a list of *entitlements*. These are `plist` files specifying the allowed degree of access to system resources and capabilities for the app [86]. Due to being embedded in the code signature, Apple has ultimate control over the specific entitlements granted to an app. Apple defines two types of entitlements on iOS [17]: (1) *restricted entitlements* that any developer account can use for any app and (2) *managed entitlements*, whose usage requires authorization by Apple for a specific app.

Intents. On Android, an *Intent* is a messaging object that can be used to request an action from another component or app. *Explicit Intents* require a specific target component. *Implicit Intents* target an abstract action type, such as viewing images. Intents can be considered part of Android's permission system insofar as they allow an app to request access to resources from another app that otherwise would require a permission. Intents do not exist on iOS, which implements item picker dialogs through API calls to UI view controllers [32], e.g., `UIDocumentPickerViewController` [31]. On both OSs, the calling app is (usually) not required to hold the respective permission, i.e., calling a document picker does not require permission to access documents. This is because the request for data is always processed directly by the user who selects the specific data to be shared with the app. However, there are exceptions on iOS, e.g., the `MPMediaPickerController` [28] requires a `NSAppleMusicUsageDescription` in the `Info.plist`.

Bind Permissions. Android lets apps implement functionality through interfaces, such as services, that can be used by other apps. For example, an app that provides a VPN has to expose a service that needs to be protected by the "`BIND_VPN_SERVICE`" permission. The permission itself is signature-protected and only available to the special "Android" built-in system package. The VPN app thus does not need to request a permission, but instead protects its VPN service by being available only to the system on behalf of the user.

2.3 Evolution of Permission Systems Over Time

Analyzing the evolution of permissions provides insights into how the protection of personal data collected, processed, and stored by a phone has changed over time on the two major mobile platforms. Permission changes not only reflect the technical evolution of smartphones but also shifting notions of privacy and data protection. Thus, we analyzed permission changes from their introduction until the latest released OS versions at the time of writing (Android 16 and iOS 26; February 2026), compiled a list of the most important privacy-related changes, and created a timeline of these milestones for both Android and iOS in Figure 1. Below, we briefly outline key developments and report a more detailed analysis in Appendix A.

The permission systems of Android and iOS began with fundamentally different approaches, which is one reason for the existing differences in permission handling described in Section 2.2. iOS introduced runtime permissions in 2008 (iPhone OS 2), app-level controls for each protected resource in 2010, and later added mandatory developer-provided permission rationales and finer-grained permission controls. Android originally employed install-time permissions in 2008 and added runtime permissions in 2015, followed shortly by finer-grained permission controls and restricted background access to multiple resources. As of 2026, the underlying technologies behind permission systems still differ widely across the two ecosystems, notably including the transparency mechanisms developers are supposed to use to make the need for specific permissions (and thus data collection) transparent to users.

Despite these conceptual differences, over time both OSs have adopted capabilities and concepts from the other platform, such as runtime permissions or the Local Network permission. Besides evolving notions of privacy, this development was fueled by legal requirements that forced Apple to take reluctant first steps towards opening up their closed-by-design iOS. For example, under the EU's Digital Markets Act (DMA) [40] from 2022, Apple had to allow

third-party apps to replace default iOS system apps, including the App Store, which required introducing new permissions.

3 Mapping iOS and Android Permissions

Although Android and iOS share similarities in their permission systems, they differ in fundamental ways. Consequently, to ensure fair, meaningful, and replicable cross-platform analyses of apps, it is vital to examine how permissions map across the two ecosystems and where permission systems are not easily comparable.

3.1 Assumptions

We base our analyses on Android Open Source Project (AOSP)-based builds of Android for Pixel devices, maintained by Google. As all licensed Android devices must pass an official compatibility test suite [68], we assume that our taxonomy is valid for the majority of the Android ecosystem. Further, we focus on the app perspective on permission systems, so we limit our evaluation to permissions that third-party developers can use, neither considering Android permissions usable only by system apps provided by the manufacturer nor any iOS APIs available only to Apple. Also out of scope are Mobile Device Management or multi-user scenarios. Both require ways to sidestep the regular permission mechanisms to allow an individual or organization to enforce behavior, such as remotely wiping a device. Finally, we focus on the OSs' intended protection mechanisms; out of scope is abuse of bugs to circumvent access to protected resources, e.g., privilege escalation.

Hence, for our taxonomy, we consider all permissions with protection level "normal" or "dangerous" available in Android 16 (API level 36, released in June 2025). On iOS, we consider all protected resources and entitlements available for third-party use and officially documented as of iOS 26.0 Beta (released in July 2025).

3.2 Permission Mapping Methodology

Since neither public resources nor related work provide a suitable method to map permissions across different mobile platforms, we propose a methodology to find and build these mappings. First, we compiled a list of all permissions available to third-party apps (i.e., not only usable by system apps) that are publicly documented in the official Android [53] and iOS [18, 29] documentations. This yielded 160 permissions on iOS (39 protected resources and 121 entitlements) and 344 permissions for Android (including permissions declared in modules). Before Android 10, permissions were declared in the main Android framework `AndroidManifest.xml`, but with Project Mainline [72], components were split into modules to help keep the system up to date. In Android 16, only health-related permissions in these modules are usable by third-party apps.

In order to create a mapping between the permissions of the two platforms, we decided to map equivalent permissions of both platforms based on their main protection target (e.g., contacts, NFC). For transparency and reproducibility, we include alternative categorization approaches that did not work in Appendix B.

To define high-level permission categories, we first extracted categories from Apple's protected resources [29] and entitlements [18] as a preliminary basis. As Apple's categorization lacks sufficient granularity, we extended and refined these categories into finer-grained categories, e.g., we split the category "Camera & Microphone" into two individual categories. To ensure that starting from the iOS perspective did not bias the mapping towards iOS concepts, we incorporated Android permission groups [55] into our categorization scheme, which only added one group, Notifications.

Table 1: Cross-Platform Permission Categories (XPPC), with descriptions and numbers of included permissions.

XPPC	Included Permissions		
Accessory Setup	Pair and manage accessories and companion devices.	7	1
Authentication	Manage and provide credentials for auto-filling in form fields.	2	1
Biometrics	Access to biometric sensors (e.g., fingerprint, FaceID).	1	1
Bluetooth	Access to Bluetooth services.	5	1
Calendar	Acces to the system (default) calendar.	2	2
Camera	Access to the device camera.	2	1
Contacts	Access to the contacts in the user's address book.	2	2
Device Management	Allows organizations to control system settings remotely.	84	1
Health	Access to the user's health-related data (e.g., heart rate).	98	8
Local Network	Access to the local network.	1	1
Location	Access to location services.	8	6
Media	Access to the user's media library (photos, videos, audio).	10	3
Messaging	Access to services related to reading and sending SMS.	6	3
Microphone	Access to the device's microphone.	2	1
Motion	Access to motion-detecting sensors.	2	2
Multicast	Allow apps to receive and send IP multicast traffic.	1	1
Networking	Allow apps to manage the network interfaces of the device.	2	1
NFC	Access to near-field communication (NFC) hardware.	4	5
Notification	Sending and processing user notifications.	1	2
Telephony Services	Access to services related to making and receiving phone calls.	15	5
VPN	Allow apps to manage virtual private network (VPN) connections.	1	1
Wi-Fi State	Access information related to the (connected) Wi-Fi network.	1	1

Next, we assigned each identified Android and iOS permission to one of these high-level categories using a qualitative coding approach. Two authors, both with strong backgrounds in Android and iOS security and privacy, independently assigned each Android permission to a group from the previously defined categories. This assignment examined the main protection targets for all grouped permissions and mapped them by access control scope, as described in the official documentation, including only permissions that provide equivalent access to the same protected resources on both OSs. In 306 (88.4%) cases, the two researchers assigned the same group. We measured inter-rater reliability using Cohen's Kappa ($\kappa = 0.86$), which indicates an "almost perfect" [89] group assignment between both coders. Where they differed in opinion, a third researcher, also with a strong security and privacy background in iOS and Android, served as a tie-breaker. Our final taxonomy maps 257 (73.2%) Android to 50 (31.3%) iOS permissions.

In the following, we refer to the individual mappings between Android permissions and their corresponding iOS permissions as *Cross-Platform Permission Categories* (XPPCs). While some XPPCs are one-to-one (1:1) mappings between iOS and Android permissions, due to differences in permission handling between platforms most are one-to-many (1:N) or many-to-one (N:1), many-to-many with equal numbers of permissions (N:N), or many-to-many with different numbers of permissions (N:M).

Taxonomy Extensions. Mobile device manufacturers can modify Android for their specific devices and introduce additional, custom permissions to support extra system features. Hence, we designed our taxonomy to be extensible and applicable to manufacturers other than Google and Apple, e.g., Samsung or Huawei. Future work looking into these manufacturers can map the additional permissions into existing categories or create suitable new XPPCs.

3.3 Permission Mapping Taxonomy

Our final taxonomy, PermiOSAn, comprises 22 XPPCs, listed in Table 1. We identified 5 1:1 mappings, 8 N:1, as well as 4 N:N and 5 N:M mappings. Table 11 in Appendix E shows the full mapping.

1:1 Mappings. For 5 categories – *Biometrics*, *Wi-Fi State*, *Multicast*, *Local Network*, and *VPN* – we could identify exactly one permission on each OS. *Biometrics* refers to an app's ability to verify the user via the device's biometric sensors, using either a fingerprint or face unlock on Android and FaceID on iOS. The other 4 XPPCs are network-related: *Multicast* enables sending multicast messages across the network, *Wi-Fi State* describes access to information about the connected Wi-Fi, *VPN* allows creating VPN connections, and *Local Network* allows scanning devices in the local network.

N:1 Mappings. In 7 cases, we found multiple permissions on Android to correspond to exactly one permission on iOS. *Authentication* on Android contains 3 different `CREDENTIAL_MANAGER_*` permissions, while iOS provides the same functionality with `autofill-credential-provider`. Similarly, for *Accessory Setup*, Android provides 7 different permissions to discover and manage accessories (i.e., `REQUEST_COMPANION_*`), while iOS 18 recently introduced a single entitlement to allow app extensions (i.e., a distinct part of an app that can be run as a service [22]) to discover and set up accessories. Another such case is *Device Management*, where Android provides 84 permissions for `MANAGE_DEVICE_POLICY_*`, while iOS only has `automated-device-enrollment.add-devices`. For *Networking*, Android's `CHANGE_NETWORK_STATE` and `CHANGE_WIFI_STATE` permissions correspond to iOS's entitlement `networking.networkextension`. The three other cases of N:1 mappings, *Bluetooth*, *Camera*, and *Microphone*, are caused by protected resources on iOS. For those, iOS governs all access and developers only have to provide rationales within the app's plist file (i.e., `NSBluetoothAlwaysUsageDescription`, `NSCameraUsageDescription`, and `NSMicrophoneUsageDescription`), whereas Android provides 5 permissions for *Bluetooth* and 2 each for *Camera* and *Microphone*. We identified one XPPC where multiple iOS permissions map to a single Android permission: *Notification*. Both systems provide a permission to receive and display them, but iOS additionally defines an entitlement for the Apple Push Notification service (APNs) [24].

N:N Mappings. We found the same amount of Android and iOS permissions for 4 XPPCs: *Calendar*, *Contacts*, *Motion*, and *Location*. For *Calendar*, Android defines read and write permissions, whereas iOS splits calendar access into full access and write-only. Similarly, Android defines read and write permissions for contacts, but iOS has one permission for general access to contacts and one entitlement for reading notes attached to a contact. In this case, the permissions on Android are finer-grained than on iOS, allowing apps to define the granularity of access they need, whereas on iOS the user decides this, e.g., by selecting which contacts to share with an app. In the *Motion* category, Android offers motion detection with the `ACTIVITY_RECOGNITION` permission and the option to request a higher sampling rate for sensors (`HIGH_SAMPLING_RATE_SENSORS`). iOS provides general motion detection (`NSMotionUsageDescription`) and fall detection (`NSFallDetectionUsageDescription`). For *Location*, both OSs define fine-grained permissions. Android's location permissions focus, e.g., on the granularity of the shared location (coarse, fine) and its source (background, media, ranging), while on iOS they are split by access modality and time (always, when in use, temporary) but also offer reduced accuracy (`NSLocationDefaultAccuracyReduced`).

N:M Mappings. We found 5 categories of N:M mappings: *Telephony Services*, *Messaging*, *Media*, *NFC*, and *Health*. *Telephony Services* and *Messaging* permissions allow third-party apps to act as the default calling, dialing and messaging app. These have been available on Android since versions 6.0 (API 23, released 2015) and 4.4 (API 19, released 2013), respectively, but iOS only recently introduced them with iOS 18 (December 2024) and iOS 26 (September 2025), which explains its current less fine-grained approach. *Media* contains permissions for accessing the user's media library (i.e., photos, videos, and audio). Here, Android distinguishes between different file types (images, video, audio), read and write access, modification of audio settings, and permissions for foreground services. On iOS, similarly to *Location*, the media library is guarded by the OS and developers can choose between access to the music library, write-only access, and full access to the media library, but ultimately the user decides which files to share. For *NFC*, Android provides permissions for general NFC access, setting the preferred payment info, and receiving NFC transactions, whereas iOS defines permissions for accessing the NFC reader, for different formats for reader sessions, and to act as the default contactless app.

In the XPPC *Health*, Android defines permissions for accessing body sensors from the foreground and background in the core system, and 95 permissions in the Mainline [72] module *Android Health*, split into read and write permissions for different data types (e.g., medical data, activity, menstruation). On iOS, `HealthKit` manages collecting and storing the user's health-related data. There are 4 `Info.plist` keys for sharing clinical health records, reading or updating `HealthKit` samples, and specifying the types of health data an app intends to access. In addition, 4 entitlements control access to Apple Health data, including required data types, background update permissions, and the ability to recalibrate health-related estimates. Compared to Android's many health-related permissions, iOS allows more flexible data types in permission requests, represented as lists in the corresponding manifests.

Permission Granularity. N:1 and N:M mappings indicate differences in the granularity of individual permissions within the same XPPC. Sometimes, one platform provides a single permission granting full access to a resource, while the other offers multiple, finer-grained permissions. This can result in over-privilege on one platform. We investigated XPPCs that request a single permission on one OS but require multiple permissions on the other. *Calendar*, *Contacts*, and *Media* allow full access on iOS, whereas on Android, there are separate read and write permissions. However, iOS also offers a read-only access permission for the calendar and allows users to restrict which contacts and photos an app can share in the permission dialog. For *Messaging* and *Telephony Services*, Android splits access into individual permissions (e.g., send and receive SMS), while iOS allows all features at once, but only for apps that replace the default messaging or telephony app. Similarly, *Notification* includes one Android permission that allows an app to display notifications to the user. On iOS, there is only an entitlement that allows apps to receive and display push notifications from remote servers (e.g., for messaging apps). The *Networking* XPPC includes

an iOS entitlement that grants apps more capabilities than the corresponding Android permissions, but it is more restricted. *Bluetooth* was covered by a full-access permission on both Android and iOS, with Android requiring the location permission for scanning. Android 12 introduced finer-grained permissions to cover connection, scanning and Bluetooth Low Energy (LE) advertising. For *NFC*, iOS offers finer-grained permissions than Android, e.g., Host-based Card Emulation (HCE) has a separate permission, which is covered by the broader `NFC` permission on Android.

Underlying Permission-Granting Mechanisms. The varying permission cardinalities on both sides of many XPPCs raise the question how and to what extent the underlying permission-granting mechanisms differ across platforms (e.g., whether the Android permissions in the *Accessory Setup* XPPC use the same permission-granting mechanisms as their iOS counterparts). We investigated these at the example of who grants a permission and compare the sets of permission-granting entities (user, OS, vendor; as indicated by the icons in Appendix E) for all permissions in each XPPC across Android and iOS (see Table 11). One third (7 out of 22) of all XPPCs share the same set of permission-granting entities. Among the remaining XPPCs, 12 (54 %) exhibit partial overlap, while the remaining three share no entities at all. The absence of overlap indicates that Android and iOS rely on entirely different strategies to protect their respective XPPCs. To quantify the degree of similarity for XPPCs with overlapping entities, we compute the Jaccard similarity [78]. Across all XPPCs, we observe an average Jaccard similarity of 0.59 (median: 0.50), indicating a moderate degree of similarity between the two platforms' permission-granting entities.

3.4 Mapping Permission-like Intents

As discussed in Section 2.2, Intents for item pickers can be considered part of Android's permission system. Their use can be estimated by disassembling the Dalvik code and searching for the corresponding constants, unless the code is obfuscated. A mapping, however, is difficult as generic Intent constants operate on collections, such as `android.intent.action.PICK` or `INSERT`. Which collection is chosen can be dynamically set in the code and the collections are managed by other apps, e.g., calendar entries by the calendar app. There exist additional limitations: `GET_CONTENT` can pick a contact but not a calendar entry. The reason is that the default calendar app on Android does not offer this interface. Another app could implement it, making generic Intent-based access a question of inter-app communication, which is out of scope.

After surveying the Intent and MediaStore documentation, we found 12 Intents that correspond directly to three different XPPCs (Media, Camera, Telephony Services). We provide a full list in Table 7 (Appendix), and eight Intent-like interfaces on iOS in Table 8. We did not investigate the usage of these interfaces on iOS for two reasons: First, it would require decrypting the apps, and second, iOS implements some Android Intents, such as `mailto:` links, only through deep links. Both topics are out of scope for our work.

3.5 Single-Platform Permissions

Permissions that are only available on one platform not only indicate fundamental architectural differences between Android and iOS but can reveal functional gaps and feature disparities between the two ecosystems. Of 344 Android permissions, 83 (24.1%) have no iOS equivalent, and the reverse applies to 111 (69.4%) iOS permissions. Tables 9 and 10 in Appendix E list all of these permissions.

Certain permissions lack a counterpart because the corresponding functionality on the other platform is not protected by a permission (e.g., basic Internet access or vibration on iOS). Some Android-only permissions grant access to properties inaccessible to apps on iOS, e.g., `READ_PHONE_STATE`. Conversely, some iOS permissions have no equivalent as Android does not provide the functionality as a core system service; instead, it defines abstract interfaces that allow third-party implementations, e.g., Google Wallet.

Other permissions only exist on one OS because there is no equivalent functionality on the other (e.g., iOS' iCloud, Journaling Suggestions, WeatherKit). Another example is Reminders, a pre-installed iOS system app for managing to-do lists and reminders. Access to these to-do list items is protected with a permission (`NSRemindersFullAccessUsageDescription`). However, there is no comparable system app on Android that allows organization of to-do lists, and hence no comparable permission.

Single-platform permissions may also reflect contrasting philosophies and policies regarding the scope of data access granted to apps. For example, Android allows apps broader access to system data than iOS. The dangerous permission `GET_ACCOUNTS` allows Android apps to retrieve a list of all accounts on the device, i.e., Google or third-party accounts. By contrast, iOS' more restrictive approach only permits an app to implement the "Sign in with Apple" feature that allows hiding the primary email address from the app and does not allow system-wide third-party accounts. As a result, developers targeting both platforms must account for asymmetries in feature availability, permission granularity, and platform governance, often requiring platform-specific design choices to achieve comparable functionality and user experience.

Although some permissions currently exist on only one of the two platforms, the other platform may introduce analogous mechanisms in future releases. Historically, both ecosystems have adopted similar approaches, as shown in Figure 1 and discussed in Appendix A. For example, at its launch in 2008, Android featured a permission system based on install-time permissions. In 2012, iOS introduced its own permission model with 7 runtime permissions. Three years later, in 2015, Android transitioned from install-time to runtime permissions. A more recent example is the Local Network permission [106], introduced by Apple in iOS 14 (2020), and followed by a comparable mechanism in Android 16 (2025).

4 Cross-Platform App Comparison

Investigating permission usage across Android and iOS helps uncover trends and inconsistencies in how permissions are adopted. In the following analysis, we apply our mapping methodology to a cross-platform dataset gathered in 2023, 2024, and 2025 to systematically examine these patterns and identify whether apps request different permissions across platforms to access (user) data.

4.1 App Pair Dataset

To evaluate the adoption of permissions and how permissions changed over time in real-world applications, we used the cross-platform app dataset from our prior work [111], containing 3,322

cross-platform app pairs of popular Android apps and their iOS counterparts, all downloaded in October 2023. Based on their identifiers from the Google Play Store and Apple App Store, we downloaded the same app pairs again in December 2024 (3,138 app pairs were still available) and in July 2025 (3,075 app pairs). The fluctuations in dataset size are due to apps having been (temporarily) removed from at least one store. For a fair comparison, we analyzed only app pairs available in all three datasets, resulting in 2,964 cross-platform app pairs, each downloaded three times between 2023 and 2025. Because some iOS features are only available to European customers (e.g., alternative app markets) and the initial dataset from 2023 was downloaded from the localized app stores for Germany, we also downloaded all apps from the German stores.

For Android apps, we extracted the declared permissions from the `<uses-permission>` and component tags in the `AndroidManifest.xml` file. For iOS apps, we extracted the permission rationales from the `Info.plist` file, as well as the entitlements from the `.entitlements` file. Sources for both Android and iOS permissions represent an upper bound, as permissions required by an app must be declared in these files but are not necessarily invoked at runtime. Further, if an app declares a permission, it can be requested by third-party components of this app, such as libraries. We do not distinguish between declared and used permissions because this is a limitation of static analysis for both Android and iOS apps, and even with dynamic analysis, it is not guaranteed that all paths that trigger permissions are covered.

As our mapping includes permissions from iOS 26 Beta (released in September 2025), these might not be reflected in our final real-world dataset (July 2025).

4.2 Permission Adoption Across Platforms

4.2.1 Observations. To better understand permission usage in practice, we analyze the XPPCs apps request in our real-world dataset. Explanations for these findings are discussed in Section 4.2.2.

Overall XPPC Prevalence. Figure 2 shows the requested permission categories on both platforms by year. Across all three years and both OSs, the most frequently used XPPCs are *Notification*, *Media*, *Camera*, and *Location*. While *Wi-Fi State* is more prevalent on Android (by 71.4%), *Media*, *Camera*, *Location*, and *Microphone* are far more often requested on iOS (by 14.5%, 34.4%, 30.0%, and 19.4%). Apps requesting the *Wi-Fi State* permission warrant closer investigation in future work. In the best case, apps use this permission to manage Internet access and reduce reliance on mobile data; in the worst case, it may be abused to infer location on older Android versions or fingerprint devices based on the network to which they are connected. One possible explanation for the differences in adoption of the *Media*, *Camera*, and *Microphone* XPPCs is that, on iOS, saving images taken with the camera requires permission to add photos to the user's media library [30]. Additionally, taking a picture with the device camera requires camera permission on iOS, whereas Android apps can use an Intent [59] for this purpose.

Overall, the prevalence of requested XPPCs between Android and iOS did not change significantly between 2023 and 2025, except for *Notification* and *Biometrics*, which were both requested more often on iOS in 2023 but on Android in 2025. Further, as shown in Table 2, for most categories the overall number of requesting apps

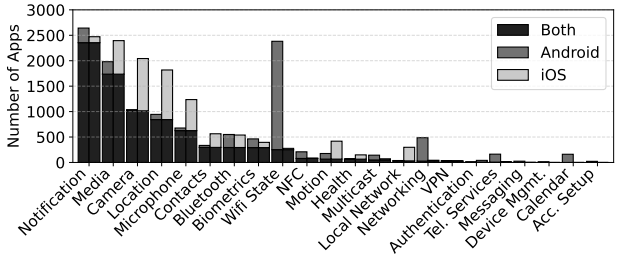

Figure 2: Requested XPPCs on both platforms in 2025. While most categories remained stable, the overall number of requested permissions has increased.

per XPPC increased over the years (on average by 72.4% on Android and 19.8% on iOS). The exceptions on Android are *Multicast*. *VPN*, and *Wi-Fi State*, where usage declined by 16.9%, 12.9%, and 0.5% respectively, while on iOS, *Networking*, *Motion*, and *Media* saw a decrease of 1.3% on average. The higher growth rate on Android suggests that during the analyzed period, permission usage across nearly all XPPCs has increased considerably faster on Android.

On Android, the number of distinct XPPCs an app requests has increased from 2023 to 2025. In 2023, around 44% of Android apps requested at least three different XPPCs, while in 2025 this number has grown to around 48%. On iOS, apps implement more XPPCs, with around 65% requesting at least 3 different XPPCs in 2023. In 2025, this number also increased slightly, but less than on Android. While iOS apps still covered a wider range of XPPCs on average, Android apps showed stronger growth in the number of implemented XPPCs, indicating that apps generally tend to request more permissions over time, potentially gaining access to more user data.

Individual Permissions Within XPPCs. We analyzed individual permissions within all XPPCs to determine what caused changes (addition/removal) in XPPC prevalence between 2023 and 2025. XPPCs with increased usage of over 100 individual permissions across Android and iOS are *Health* (549, +88.7%), *Notification* (332, +6.9%), *Location* (271, +6.1%), *Microphone* (262, +14.8%), *Biometrics* (176, +25.8%), *Bluetooth* (127, +6.5%), *Media* (122, +1.3%), and *Camera* (112, +3.7%). The increase in *Health* mainly reflects adoption of the Android Health Mainline module permissions (89.8%). In *Notification*, the Android permission `POST_NOTIFICATIONS` accounts for 99.4% of the increase, and in *Location*, `FOREGROUND_SERVICE_LOCATION` caused the largest increase (43.2%), followed by the iOS permission `NSLocationDefaultAccuracyReduced` (18.9%). Most changes in *Microphone* stem from the Android permissions `FOREGROUND_SERVICE_MICROPHONE` (39.3%) and `RECORD_AUDIO` (34.4%). The majority of the increase in *Biometrics* corresponds to `USE_BIOMETRICS` (87.5%). For *Bluetooth*, Android permissions account for 81.1% of the total increase, with `BLUETOOTH_CONNECT` alone contributing 32.3%. *Media* also changed most on Android, i.e., `FOREGROUND_SERVICE_MEDIA_PLAYBACK` (+250), `READ_MEDIA_VISUAL_USER_SELECTED` (+176), and `FOREGROUND_SERVICE_MEDIA_PROJECTION` (+64), but decreased in `READ_MEDIA_IMAGES` (-276), `READ_MEDIA_VIDEO` (-159), and `READ_MEDIA_AUDIO` (-108). Most of the increase in *Camera* stems from the Android permissions `RECORD_AUDIO`, `CAMERA`, and `FOREGROUND_SERVICE_CAMERA`, which account for 44.6%, 23.8%, and 18.3% of the total, respectively. Overall, these XPPCs increased most on Android, while iOS grew more slowly.

Coarser-grained permission bundling on one platform may lead to over-privilege. For example, Android splits calendar access into separate read and write permissions, while iOS offers read-only and full access. Thus, apps requiring only write access also gain read access on iOS. We investigated individual permissions of XPPCs, which include permissions that potentially allow over-privilege on one OS and their adoption across both platforms within our 2025 dataset, i.e., *Calendar*, *Contacts*, *Media*, *Messaging*, *Telephony Services*, *Notification*, *Networking*, *Bluetooth*, and *NFC*. Although iOS allows full calendar access, no app in our dataset requests it. On Android, both read (5.2%) and write (4.7%) access are prevalent. Full contact access is requested by 19.0% of iOS apps, while 11.3% of Android apps request read access and only 2.0% request write access. On iOS, 78.1% of apps request full photo-library access and 47.0% request write-only access. On Android, *Media* permissions are finer-grained, and requests are split across individual permissions. The most common are read (54.2%) and write (55.9%) access to external storage, with similar usage rates. For *Messaging* and *Telephony Services*, iOS grants more privileges with a single permission. However, the requirements are also very strict, and we observed only one app with the entitlement to replace the default messaging app and three apps capable of replacing the default calling app. Slightly more Android apps request notification access (89.1%) than iOS apps (83.4%). Android apps also use *Networking* permissions more often, with 11.6% requesting `CHANGE_WIFI_STATE` and 10.9% `CHANGE_NETWORK_STATE`, while only 1.5% of iOS apps adopt the network-extension entitlement, which offers more capabilities but requires an additional location permission. Although Android deprecated its broader `BLUETOOTH` permission, 18.2% of apps still use it in 2025, the same share as on iOS. Finer-grained permissions for connecting (12.8%), scanning (8.0%), and Bluetooth LE advertising (2.4%) are used less often. NFC is more frequently adopted on Android (7.1%) than on iOS (2.8%), where only 2 apps also request the entitlement for HCE capabilities. Thus, although iOS more often offers full resource access than Android, apps do not always request it. iOS also lets users further restrict access by selecting specific resources to share, e.g., individual photos. Finer-grained permissions, however, let developers request only the minimum required access, improving user privacy and making it harder for libraries or other apps to piggyback on granted permissions.

Differences Between Platforms. We investigated which XPPCs differed most in adoption across platforms. Table 3 shows that the five XPPCs with the greatest differences remained consistent over the years: *Wi-Fi State*, *Location*, *Camera*, *Media*, and *Microphone*. While *Wi-Fi State* is requested almost exclusively on Android, most other differences stem from permissions requested only on iOS. To better understand these disparities, we identified the primarily responsible permissions. For *Wi-Fi State*, `ACCESS_WIFI_STATE` is requested by 2,130 (89%) Android apps. For *Location*, the higher iOS adoption is largely driven by `NSLocationWhenInUseUsageDescription`, which appears in 965 (53%) apps. For *Camera*, `NSCameraUsageDescription` is present in 1,028 (50%) iOS apps, for *Media*, the most frequently requested iOS permission is `NSPhotoLibraryUsageDescription`, and for *Microphone*, `NSMicrophoneUsageDescription` could be observed in 610 (49%) iOS apps.

Table 2: Overall adoption growth rates per XPPC from 2023 to 2025, including total differences between Android and iOS. On Android, all but two categories showed increased adoption, with the largest growth in *Notification* and *Biometrics*. On iOS, most categories increased only modestly, with the greatest gains in *Microphone*, *Local Network*, and *NFC*.

	2023		2025		Growth (%)		Diff.
							–
Notification	2312	2470	2642	2472	330 (14.3)	2 (0.1)	328
Biometrics	309	373	463	395	154 (49.8)	22 (5.9)	132
Networking	428	44	487	43	59 (13.8)	-1 (-2.3)	60
Media	1925	2398	1981	2395	56 (2.9)	-3 (-0.1)	59
Health	17	140	68	149	51 (300.0)	9 (6.4)	42
Location	920	1818	945	1818	25 (2.7)	0 (0.0)	25
Microphone	585	1167	677	1236	92 (15.7)	69 (5.9)	23
Telephony Services	133	3	163	11	30 (22.6)	8 (266.7)	22
Camera	990	2016	1038	2043	48 (4.8)	27 (1.3)	21
Calendar	144	0	161	0	17 (11.8)	0 (0.0)	17
Motion	167	424	176	418	9 (5.4)	-6 (-1.4)	15
Accessory Setup	16	0	22	0	6 (37.5)	0 (0.0)	6
Messaging	20	0	25	1	5 (25.0)	1 (new)	4
Device Management	10	0	13	0	3 (30.0)	0 (0.0)	3
Contacts	328	560	335	564	7 (2.1)	4 (0.7)	3
VPN	35	28	34	29	-1 (-2.9)	1 (3.6)	-2
Authentication	7	35	11	41	4 (57.1)	6 (17.1)	-2
NFC	181	52	209	84	28 (15.5)	32 (61.5)	-4
Bluetooth	539	515	550	539	11 (2.0)	24 (4.7)	-13
Local Network	27	259	38	298	11 (40.7)	39 (15.1)	-28
Wifi State	2395	258	2382	277	-13 (-0.5)	19 (7.4)	-32
Multicast	172	68	143	72	-29 (-16.9)	4 (5.9)	-33

Evolution of Differences Across Platforms. Next, we explored how the differences in permissions between Android and iOS evolved over the years. Figure 3 illustrates how frequently apps have included or removed an XPPC between 2023 and 2025, as well as the resulting net total for each category. *Wi-Fi State*, *VPN*, and *Multicast* were the only categories with reduced usage, while *Biometrics*, *Notification*, and *Microphone* showed the highest growth. Our results show that requested permissions fluctuate, but overall, most XPPCs exhibited increased usage from 2023 to 2025.

To evaluate the impact of newly introduced permissions on the increase of permission usage over time, we examined recent Android and iOS releases. Android 16 (June 2025) introduced only 3 permissions in our cross-platform taxonomy: `RANGING`, which allows apps to measure distances between devices, and 2 device management-related permissions; Android 15 (September 2024) contributed 6 device management-related permissions. None are used by any app in the dataset. The permissions introduced with Android 14 (October 2023) caused only small increases in the use of specific XPPCs: +52 apps in *Media*, +8 in *Health*, and +2 in *Microphone*. Since they almost always occur together with other permissions from the same XPPC, their overall impact remains negligible. Despite iOS 17 introducing 26 new permissions in September 2023, only 5 are covered by our taxonomy. Of these, merely 3 were observed in our 2025 dataset, all related to *NFC*, with usage ranging from 1 to 5 apps. iOS 18 (September 2024) added 14 new permissions, 4 of which are included in our taxonomy, yet none appear in the 2025 dataset. This shows that the increase in permission usage on Android and iOS cannot be attributed to newly introduced permissions but rather reflects broader changes in how apps request existing permissions.

Next, we examined whether apps generally used more permissions or simply replaced some of them, by assessing changes in the average number of XPPCs from 2023 to 2025 on both OSs. In 2023, Android apps requested on average 3.9 XPPCs, compared to 4.3 for iOS apps. By 2025, the Android average had increased to 4.2,

Table 3: Differences in adoption rates for the top 5 XPPCs used exclusively by one app in a pair. On Android, usage increased across all 5 categories, while on iOS usage declined.

	2023			2024			2025		
	Total	(%)	(%)	Total	(%)	(%)	Total	(%)	(%)
Wi-Fi State	2189	2163 (98.8)	26 (1.2)	2158	2132 (98.8)	26 (1.2)	2155	2130 (98.8)	25 (1.2)
Location	1082	92 (8.5)	990 (91.5)	1082	94 (8.7)	988 (91.3)	1077	102 (9.5)	975 (90.5)
Camera	1028	72 (7.0)	956 (93.0)	1020	76 (7.5)	944 (92.5)	990	72 (7.3)	918 (92.7)
Media	897	212 (23.6)	685 (76.4)	914	257 (28.1)	657 (71.9)	904	245 (27.1)	659 (72.9)
Microphone	658	38 (5.8)	620 (94.2)	669	44 (6.6)	625 (93.4)	661	51 (7.7)	610 (92.3)

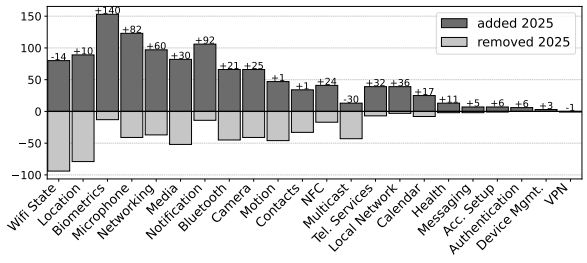

Figure 3: Number of XPPCs added or removed between 2023 and 2025. Some XPPCs are used less (e.g., *Wi-Fi State*, *Location*), but apps tend to request more XPPCs over time.

while iOS remained at 4.3. We also analyzed the average number of XPPCs requested per Apple App Store category for all three years on both platforms (see Table 6). The highest averages occurred in Social Networking, Lifestyle, Business, Travel, Finance, Navigation, and Utilities. In 2025, those categories averaged at least 6 different XPPCs on both platforms, with Social Networking even reaching 7 on Android. In 2023, iOS apps requested more distinct XPPCs than Android apps in 29 Apple App Store categories (74.4%). By 2025, nearly all categories saw increases on Android, narrowing the gap and even leading to higher numbers than on iOS in 17 categories (43.6%). This suggests requested XPPCs on Android have grown faster than on iOS, with variation across app categories. Fewer XPPCs on one OS may indicate the presence of services rather than permissions. For example, iOS apps that rely exclusively on system-managed AirPlay interfaces to stream content to devices on the local network do not need to request the local network permission, as device discovery and communication are handled by iOS [106].

Differences in Cross-Platform App Pairs. We analyzed differences in the adoption of XPPCs between Android and iOS app pairs for all three years. For each app pair, we counted the number of XPPCs present on only one platform: around 61% differ by at most three XPPCs, while fewer than 10% differ by more than five. Although individual apps change their requested XPPCs, year-to-year variance in permission differences is low, indicating overall stability. These results also suggest that substantial differences in requested permissions reflect functional differences across platforms.

Figure 5 (Appendix) shows the number of requested XPPCs of all app pairs on Android and iOS for 2023 and 2025. The difference in the number of XPPCs between app pairs is, on average, between 3 to 4, but such differences occur quite frequently. However, we found that 2,916 (98.4%) app pairs exhibited at least one difference in XPPC adoption across platforms over the years. This indicates platform-specific behavior within the same apps. In 2023, 76 (2.6%) app pairs showed identical XPPCs across both platforms, increasing to 82 (2.8%) in 2024 and to 89 (3.0%) in 2025. A few outliers also request noticeably more permissions on only one platform.

4.2.2 Reasons for Diverging Permissions. To understand the differing permissions for the same app across platforms, we manually investigated apps with different XPPCs on Android and iOS.

Permissions of Third-Party Components. One major factor behind the differences are the design philosophies behind permission declarations. On Android, if a permission is not requested, accessing resources returns no results or, at worst, crashes the app. If a third-party library is added, the required permissions can either be automatically added to the manifest or omitted. On iOS, however, developers are required to add all permissions of resources that are referenced in the code, even if the code is never executed [20]. An example for such a case is the React Native library "react-native-permissions" [1], which implements cross-platform permission handling. Prior to version 2.0.0 (September 2018), all permissions, even if not used, had to be declared in the `Info.plist` file [2]. We found 30 apps in 2023 and 10 apps in 2025 explicitly stating this library in permission usage descriptions. We also found other apps that declare a permission and justify it as required by a third-party library or API (24 apps in 2025) or not being used by the app at all, without further explanation (90 apps in 2025). Apple's App Store review guidelines are a major factor in permission divergence between platforms, leading to forced over-permissioning and potentially exposing apps to risks such as permission piggybacking.

Companion Apps. Companion devices like smartwatches offer diverse functionality, e.g., forwarding calls and messages from the phone to the device, and corresponding apps therefore tend to request multiple different permissions. We found that companion apps tend to request very different XPPCs on Android than on iOS. They typically request *Telephony Services*, *Messaging*, *Calendar*, and *Accessory Setup* only on Android, and *Bluetooth*, *Camera*, *Microphone*, and *Notification* on iOS; examples include Samsung Galaxy Fit (Gear Fit), Samsung Galaxy Watch (Gear S), and Garmin Connect. This is rooted in the platforms' different approaches towards companion devices. On Android, third-party companion devices are well integrated and, coupled with their corresponding app, can receive messages and phone calls and essentially forward functionality from the phone. On iOS, integrating third-party devices is not straightforward, as these devices are essentially paired as Bluetooth accessories and thus cannot forward phone calls or messages. With iOS 26.3, Apple first added notification forwarding functionality, available only in the EU due to the DMA, and subsequently released the Accessory Notifications framework in iOS 26.5 [21].

Keyboard Apps. Keyboard apps are another category with substantial differences between Android and iOS. On Android, these apps can be just a keyboard selectable via the systemś input settings. On iOS, apps that define a keyboard extension [26] always include a normal app component. We found that apps tend to use this to provide additional functionality on iOS. For example, "Grammarly: AI Writing Keyboard" requests *Camera*, *Location*, and *Media* access only on iOS because it also provides a document-writing assistant in the iOS app, allowing users to scan and import documents and photos, whereas the Android app only provides keyboard settings.

Removal of *_MEDIA_* Permissions. From 2023 to 2025, across all analyzed Android apps, we observed a decrease in the use of the READ_MEDIA_IMAGES (-37%) and READ_MEDIA_VIDEO (-35%) permissions, which regulate access to the phone's photo gallery. This stark decrease can be attributed to Google's initiative to restrict broad access to the photo gallery to increase users' privacy [65], announced in 2023 and enforced in 2025. Google proposes using the "Android photo picker" as a privacy-friendly alternative, as the app does not need to handle permissions to access the media store [47]. Apps might try to circumvent this by requesting additional permissions to access media storage. In our dataset, 370 apps (12.5%) removed *_MEDIA_* permissions but added other permissions to access media files (e.g., READ_EXTERNAL_STORAGE, READ_MEDIA_VISUAL_USER_SELECTED). This shows that streamlining processes and designing them in a more privacy-friendly way can impact, at scale, the personal data that apps might access.

Signal. In 2023, Signal requested 6 XPPCs only on Android (*Bluetooth*, *Calendar*, *Telephony Services*, *Messaging*, *Networking*, *Wifi State*). Messaging was removed in February 2024 [110] after Signal had announced discontinuing SMS support in October 2022 [108] and removed the *Calendar* and other unused permissions in March 2024 [109], leaving 4 XPPCs by 2025. This shows that one source of disappearing permissions may be developers becoming aware of over-permissioning and subsequently addressing it.

Reasons for Diverging Permissions For Same Apps . During analysis we identified several benign causes of permission divergence within the same app across platforms. Most prominent are differences in the capabilities of Android and iOS, such as permission models (e.g., Wi-Fi State), platform-specific APIs (e.g., Media), and, especially on Android, backward compatibility with older OS versions. Another reason for diverging permissions are libraries or SDKs. On iOS, every potential resource access must be covered by permissions in accordance with Apple's Review Policies, which also apply to libraries. Here, iOS prioritizes app stability, crashing an app if tries to access a resource without permission (i.e., missing usage description), whereas Android apps do not necessarily crash when accessing unauthorized resources. Apps may also implement different functionality across platforms, e.g., due to different development teams or companies. We further investigated reasons for removing permissions and identified the following: the introduction of privacy APIs like picker dialogs (e.g., photo picker), platform policy changes, removal of libraries or SDKs, unused permissions, legacy code, or removal of permissions/capabilities due to negative user feedback. There can also be malicious or privacy-invasive reasons for permission divergence. One example are third-party libraries or SDKs trying to collect data by probing for permissions of the main app, such as the advertising library Adjust [3] checking for ACCESS_WIFI_STATE to access the MAC address of the user's device. Other possibilities not observed in our dataset are hidden device fingerprinting, collection of more data on one OS due to less strict permissions, or inherent over-permissioning on the other OS.

Trends and Policy Changes. We observed that permission usage grew faster on Android than on iOS from 2023 to 2025. Some changes can be attributed to policy changes of Android and the Google Play Store. Android 13 (2022) introduced the POST_NOTIFICATION permission, explaining the large increase (+330) in 2025, as apps continued to adopt it. Although this increases the number of declared permissions, it strengthens user privacy by giving users explicit control over whether applications can send notifications. The permissions READ_MEDIA_IMAGES, READ_MEDIA_VIDEO and READ_MEDIA_AUDIO were initially introduced in Android 13 to replace READ_EXTERNAL_STORAGE [53]. In October 2023 [56, 73], however, Google announced that these media permissions may only be used if system pickers cannot provide sufficient functionality, effective January 2025. Hence, we observed a shift from those media permissions to the picker-related permission READ_MEDIA_VISUAL_USER_SELECTED. This enhances user privacy by allowing users to share only selected images or videos rather than their entire media library. Android 14 (2023) introduced stricter requirements for foreground services, i.e., apps had to declare explicit service types [70] and request the corresponding permission. This improved transparency by clarifying the purpose of each service, enforcing appropriate permissions and runtime requirements for sensitive resources, and preventing misuse of foreground services for unrestricted background execution [71]. We observed a strong increase in foreground services-related permissions (+613) in 2025.

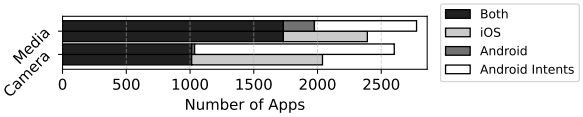

Figure 4: XPPC differences with Android Intent usage. iOS requires apps to explicitly declare camera access. On Android, apps can request media capture via an Intent, but access to the captured media requires explicit user action.

4.3 Intents

We estimated the usage of Intents to access Media, Camera, and Telephony Services by disassembling all Android apps with dexdump (Android build tools v37.0). All but 17 apps were successfully analyzed this way, and searched for constants typically necessary to use the Intents. Figure 4 shows how many apps reference Intent constants but do not request the related permissions. No app used the Telephony Services Intent, so we excluded it from the plot. This alternate way of accessing resources can explain the gaps: 1,566 apps use Intent-based Camera access on Android without requiring a related permission, and 801 do the same for Media access. Unlike permissions, this estimate provides neither an upper bound (because the code might be obfuscated) nor a lower bound (because the code might never be called).

4.4 Android-only Permissions

In our dataset, we identified 10 Android-only permissions used by at least 10% of the analyzed apps as of 2025, as shown in Table 9. Table 9 in Appendix D.2 shows the full list of Android-only permissions.

Networking. The most commonly used of these are INTERNET and ACCESS_NETWORK_STATE, which are present in nearly every app in our dataset and are automatically granted at installation (protection level normal). These permissions do not have direct iOS counterparts because iOS does not require developers to declare network-related capabilities in the app manifest. Basic outbound

networking, covered by INTERNET on Android, is permitted by default for all apps on iOS. Likewise, high-level information about the network state, governed by ACCESS_NETWORK_STATE on Android, can be obtained on iOS via APIs such as NWPathMonitor [19] without requiring a permission.

Job Service. Android uses the BIND_JOB_SERVICE permission to bind to an app's JobService, allowing the app to schedule tasks or background executions. iOS provides background task scheduling through system APIs without requiring an equivalent permission.

Phone State. Around 24% of Android apps in our dataset use the READ_PHONE_STATE permission, which is classified as dangerous and allows apps to read information about the current cellular network, the status of ongoing calls, and registered PhoneAccounts.

Accounts. GET_ACCOUNTS allows to access all accounts registered with the system's Account Manager, e.g., email. On iOS, by contrast, apps cannot enumerate the user's system-wide accounts.

Foreground Service. FOREGROUND_SERVICE allows Android apps to elevate a service's priority, thereby protecting it from termination by the OS under resource pressure or while the user interacts with another app. Since Android 14 (API 34), this requires an additional service type declaration, such as camera or dataSync. For example, the FOREGROUND_SERVICE_DATA_SYNC permission enables an app to run a dataSync foreground service for tasks such as data up- and download. The documentation recommends using purpose-built APIs where possible. There is also a specialUse type in case none of the pre-defined ones apply [70], but it requires additional justification and review during app submission to the Google Play Store. An equivalent customizable foreground service does not exist on iOS, which enforces stricter resource and user-control policies. iOS enforces a segmented, purpose-built approach that allows only certain categories of apps to run continuously (e.g., location updates, VoIP, external accessory communication, Bluetooth LE accessories, and background fetch and processing. General background work is subject to system-wide scheduling to protect battery life.

Boot Completion. The permission RECEIVE_BOOT_COMPLETED, found in more than half of Android apps, guards access to a broadcast notifying apps about system boot completion, allowing them to install services before they are started by the user. On iOS, the app life cycle allows an app to be notified when it is started, but not when the device finishes booting or is shutting down.

Wake Lock. Another highly used Android-only permission that we observed in 99.7% of apps is WAKE_LOCK, which allows apps to prevent the device from entering an idle state by keeping the CPU running or the screen turned on. On iOS, there is no comparable permission, and apps may only disable the device's idle timer while they are in the foreground. Under this approach apps can only prevent the screen from turning off but cannot keep the CPU running in an active state. Once an app moves to the background or the user turns off the screen, the system may suspend or terminate the app.

Vibration. In over 70% of the Android apps in our dataset we found the VIBRATE permission, which allows apps to vibrate the device and is automatically granted at installation time. No comparable permission exists for iOS, where apps can invoke vibration through APIs and only while they are running in the foreground.

Table 4: Usage of Android-only permissions in the 2025 dataset (used by at least 10% of apps) that have no iOS counterpart (e.g., no INTERNET permission equivalent on iOS).

Permission	Protection Level	Total	%
INTERNET	normal	2,960	99.87
ACCESS_NETWORK_STATE	normal	2,960	99.87
BIND_JOB_SERVICE	signature (bind)	2,890	97.50
WAKE_LOCK	normal	2,880	97.17
FOREGROUND_SERVICE	normal	2,670	90.08
VIBRATE	normal	2,098	70.78
RECEIVE_BOOT_COMPLETED	normal	1,517	51.18
READ_PHONE_STATE	dangerous	701	23.65
FOREGROUND_SERVICE_DATA_SYNC	normal, instant	618	20.85
GET_ACCOUNTS	dangerous	331	11.17

Other. Of the remaining 66 Android-only permissions in our dataset – all used in fewer than 10% of apps – 17 (25.8%) are deprecated. The others are Android-specific features that are either unavailable to third-party apps on iOS (e.g., SET_WALLPAPER) or not permission-protected on iOS (e.g., DETECT_SCREEN_RECORDING). Most (68.2%) relate to app life-cycle and process management, niche telephony frameworks, UI overlay controls, device personalization, alarms and scheduling, foreground services, bind permissions, and the "sync adapter" [69]. Some (7.6%) concern security features, security policies, and detection of screen captures and recordings.

4.5 iOS-only Permissions

We found 11 iOS-specific permissions used by at least 10% of the analyzed apps from 2025, presented in Table 5 and described below. Table 10 in Appendix D.3 shows all iOS-only permissions.

Tracking. We observed the iOS-only permission NSUserTrackingUsageDescription in 74.73% of apps in our dataset. It is associated with AppTrackingTransparency (ATT) [23] and must be present if an app engages in user tracking or accesses the device's advertising identifier (AdID). On Android, access to the AdID is managed through the AdServices API (SDK Extension), which defines the relevant permissions. However, this API is part of Google's Privacy Sandbox, which was phased out in October 2025 [67]. Otherwise, apps can obtain the AdID through the AdvertisingIdClient, which is provided by Google Play Services and protected by the custom permission com.google.android.gms.permission.AD_ID.

Interprocess Communication (IPC). The com.apple.security.application-groups entitlement enables apps from the same developer to use IPC, read and write data in shared containers, and access shared keychain items. Beyond this mechanism, iOS does not provide an official IPC channel between apps due to its sandboxing model and security constraints. By contrast, Android offers multiple IPC mechanisms that allow apps to communicate without sharing a developer, such as Binder and Intents.

Keychain. The keychain-access-groups entitlement works similarly to the app-groups entitlement: It allows apps from the same developer to share access to items in the keychain, functioning more as a development configuration than a traditional permission.

Encryption. The most frequent iOS-only permission, ITSAppUsesNonExemptEncryption, uses a boolean value indicating whether the app uses encryption that may fall under an encryption export regulation [25]. Interestingly, this key is documented only for macOS, not for iOS, which is also reflected in our data: 92.4% of iOS apps that define this plist key set it to false. However, developers

Table 5: Usage of iOS-only permissions in the 2025 dataset (used by at least 10% of apps) which are designed for use cases that are not provided by Android (e.g., encryption export regulation) or not protected by a permission (e.g., deep links).

Permission	Total	%
ITSAppUsesNonExemptEncryption	2,300	77.60
NSUserTrackingUsageDescription	2,215	74.73
com.apple.developer.associated-domains	1,929	65.08
com.apple.developer.applesignin	1,453	49.02
com.apple.security.application-groups	1,233	41.60
UIRequiresPersistentWiFi	1,217	41.06
keychain-access-groups	1,081	36.47
com.apple.developer.game-center	659	22.23
com.apple.developer.icloud-container-identifiers	410	13.83
com.apple.developer.ubiquity-kvstore-identifier	347	11.71
com.apple.developer.icloud-services	297	10.02

have reported that the TestFlight status of their apps displays a warning if this key is not set [42].

Deep Links. `com.apple.developer.associated-domains` can be used to specify associated domains for deep links, a functionality that Android apps implement through Intent filters instead.

Persistent Wi-Fi. The plist key `UIRequiresPersistentWiFi` can be used to specify that an app requires a persistent Wi-Fi connection while it is running, as iOS otherwise closes inactive Wi-Fi connections after 30 minutes.

Apple Services. `com.apple.developer.applesignin` provides support for "Sign in with Apple". The `com.apple.developer.icloud-services` entitlement enables access to both CloudKit and iCloud Documents, while `com.apple.developer.icloud-container-identifiers` and `com.apple.developer.ubiquity-kvstore-identifier` each allow apps to store data (CloudKit) or documents (iCloud Documents). The `com.apple.developer.game-center` entitlement enables Game Center features in an app, e.g., achievements or leaderboards. Android offers similar functionality through Play Games Services, without requiring a permission.

Other. Our dataset contains 102 iOS-only permissions with less than 10% usage: 85 entitlements and 17 usage descriptions. Most (37.3%) relate to Apple-specific services and frameworks, such as iCloud, HomeKit, Apple Pay, WeatherKit, App Attest, and Siri. Others are entitlements that are development-specific, allow apps to restrict content (e.g., family control) and moderate notifications, define app extensions (media, camera, networking), means for inter-app communication, memory and performance controls, and advanced communication features, e.g., VoIP and device pairing. We found 7 deprecated permissions, 6 documented only for macOS, and 1 documented only for tvOS. We also found 5 private entitlements: one is used by 11 third-party gaming apps, and the other 4 are all used by Shazam, which is distributed by Apple itself.

5 Recommendations

Developers. Our results show that mapping permissions between platforms is non-trivial. Both Android and iOS handle permissions only at the app level, not at the code component level. As modern app development heavily relies on third-party libraries, the burden of verifying their required permissions falls on developers. Cross-platform frameworks (e.g., React Native [98]) provide only minimal support for cross-platform permission management (see Appendix B), leaving developers to configure them by hand, which is error-prone and time-consuming.

Cross-platform development frameworks can use our results to develop tooling that automatically generates a (minimal) set of permissions for both platforms, thus resulting in apps that use a minimal, comparable set of permissions by default. Teams who develop the same app for a single or both ecosystems can benefit from our mapping to check if both versions of the app request a similar set of permissions and, if necessary, address over-permissioning.

OS Vendors. Our results yield two insights into how OS vendors can adapt permission system designs to reduce systemic over-permissioning across both analyzed platforms. On iOS, Apple's review process requires that apps include permission strings for any API referenced in their code, even if never invoked at runtime. This is particularly problematic when integrating large monolithic third-party libraries for a subset of their functionality. As a possible remediation, a scheme similar to Android's could be adopted, where the developer is trusted with the declared permissions, and any undeclared runtime request fails. On Android, the user can only grant runtime permissions on a per-group basis. As a result, an app can silently access data or functionality protected by a different permission in a permission group for which the user consciously granted access. As a remediation, we recommend adding more fine-grained permission control. To retain existing UI flows, fine-grained controls could initially be added only to the system settings (not to the permission dialog) or as an optional advanced mode.

Market Distributors. Markets and app distributors can use our mapping to verify that the same app does not use vastly different permissions on different platforms. The mapping can also help app store providers automatically derive *privacy labels* [4, 116] based on an app's used permissions. Transparent app metadata that is comparable across platforms can help increase user trust or foster cross-platform alternative markets. We recommend a balance between security and openness; e.g., limiting the `targetSdkLevel` can help prevent apps from abusing backwards compatibility.

Security and Privacy Researchers. Prior comparative cross-platform app analyses were limited by focusing on only a subset of permissions across platforms, offering limited insights into apps' security and privacy practices. Our taxonomy allows to compare a comprehensive set of permissions between platforms, taking into account fundamental differences in permission systems. This lets researchers conduct cross-platform analyses that draw fair comparisons between the permissions requested by apps on each OS.

End Users. For users, platform differences in permissions affect what apps can do. While Android offers a plethora of APIs, iOS is more selective about what an app can do and what information it can access. Our permission mapping and categorization enables users to compare (favorite) apps across both OSs to make more informed decisions before purchasing a new device. It also allows for a first assessment of whether apps respect user privacy, such as the data minimization principle (Art. 5 (c) GDPR [43]).

Regulators and Privacy Advocates. Our taxonomy can support regulators, digital rights organizations, and privacy advocates with finding concrete cases of potential privacy law violations to investigate. Our mapping can help identify suspicious apps that require different permission categories across platforms, which *could* indicate privacy violations or over-permissioning and can then be developed further into concrete cases of privacy law violations.

Regulators can use the discrepancies highlighted by the mapping to identify trends in permission usage, which in turn could inform decisions about new pathways for regulation. Further, the mapping can uncover (unfair) platform-specific practices such as not offering certain features to third parties to gain an advantage that is not compatible with current law (e.g., the EU's Digital Markets Act). Privacy advocates can use the mapping to easily identify app pairs that differ significantly in permissions, which they can then use to build cases that highlight privacy issues in mobile ecosystems.

6 Limitations and Future Work

Our PermiOSAn taxonomy currently does not cover the full set of permissions available on Android and iOS. We only analyzed the Android version used by Google Pixel phones, and therefore exclude permissions defined by Android derivatives.Still, the analyzed permissions are a subset found on all Google-certified Android devices, so we assume our key findings are valid for the majority of the ecosystem. We excluded all permissions specific to individual Google services—except the highly privacy-relevant health-related permissions—to better reflect the permissions available on most Android smartphones. Future work can extend our taxonomy to excluded permissions, other vendors' Android versions, and future updates to the iOS and Android permission systems. Our analysis covers permissions up to Android 16 and iOS 26 Beta, the most recent OS versions at the time of writing (February 2026). We only analyzed declared permissions, as actual permission usage cannot be reliably inferred from static analysis alone and may not be fully observable even with dynamic analysis due to limited path coverage.

7 Related Work

App permissions are a widely used metric for studying the security and privacy implications of mobile apps and identifying potential overreach. Thus, over the years, a large yet fragmented body of knowledge has been created about app permissions. While there are some insights about the prevalence and evolution of permissions, only very few works investigated iOS, let alone made an effort at aligning selected permissions across platforms.

App Permissions in Security & Privacy Research. A prominent line of research keeps showing that many apps are *over-privileged*, i.e., request more permissions than what they would need for their functionality [44, 81]; one potential reason is insufficient documentation [44]. Common approaches to identify unnecessary permissions include analysis of app descriptions [99] and determining apps' "typical" functionality or frequent permission patterns [5, 105, 115]. Other work showed that permission mechanics can be bypassed through app updates [39], custom permissions [87], or side and covert channels [100]. Research in usable security and privacy has studied app permissions from the perspectives of end users [6, 37, 38, 45, 77, 82, 92, 97, 106, 107] and developers [112], with both groups exhibiting knowledge gaps about the necessity, scope, and implications of permissions and associated data flows.

Empirical Measurements of App Permissions. Prior work has studied the evolution and usage of app permissions over time for Android. A first longitudinal study was conducted by Wei et al. [114] on data from April 2009 to December 2011. They found that the number of permissions used by apps grew to comprise additional hardware features. The investigated set of 346 pre-installed and 237 third-party apps was found to routinely request more permissions than necessary for their functionality. Almomani and Al Khayer [6] comprehensively studied the mechanisms and evolution of the Android permission system from 2008 to 2022, identifying continuous growth in the number of permissions per permission category over time; this trend was paralleled by apps adopting increasingly larger numbers of permissions. Our findings show that this trend continues into 2023–2025. In 2024, Tuncay [113] reported on Android permissions from the perspective of Google, outlining the evolution of the permission ecosystem with the underlying rationales, permission statistics over time, and best practices for developers. Concurrent to our work, Alkinoon et al. [5] investigated how Android permissions evolved between 2019–2023, with a focus on differences between malicious and benign apps.

While permission usage prevalence and patterns of Android apps have been well studied [35, 81], measurements of app permissions on iOS are scarce. Outside of academia, Apple device management company JAMF reported for data from Q3/2021 that the most common permissions requested by the investigated 100K App Store apps were Photos, Camera, Location, and Microphone [79].

Categorizing and Cross-Platform Mapping of Permissions. Prior cross-platform work has identified the need for mappings between concepts of the iOS and Android ecosystems to facilitate meaningful comparison [85, 111]. For example, in their cross-platform privacy comparison of iOS and Android apps, Kollnig et al. identified a total of seven permissions that exist on both systems and were thus deemed "particularly dangerous" enough by both vendors to require opt-in [85]. Alkinoon et al. suggested higher-level categories for Android permissions that were supposed to facilitate subsequent analysis but did not revisit these later [5]. Khandelwal et al. mapped the data types in Apple's App Privacy Labels to those in the Google Play Store's Data Safety Sections [83], and Arkalakis et al. [34] created a mapping between 17 Google Play Data Safety Sections and 94 API methods that the investigated Android apps called to access the corresponding information.

Our work goes beyond these initial, limited-scope efforts and proposes a mapping of Android and iOS permissions to make future cross-platform analyses more comparable and meaningful.

8 Conclusion

In this work, we developed PermiOSAn, a taxonomy of cross-platform permissions that comprises 22 Cross-Platform Permission Categories (XPPCs), mapping 257 (73.2%) Android permissions to 50 (31%) iOS permissions. This taxonomy provides a foundation for future studies to compare apps across platforms on an app-by-app basis. Applying our taxonomy to 2,964 cross-platform app pairs from 2023–2025, we showed that (1) iOS apps request more XPPCs, while Android apps exhibited a marked increase in the number of requested XPPCs and the adoption rate of almost all individual XPPCs, (2) divergence in the adoption of permissions across platforms has different reasons, such as differences in platform-capabilities, and apps offering different functionalities, and (3) changes in the adoption of XPPCs are largely caused by policy changes, and apps removing individual features, libraries or legacy code.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback to improve the paper. This work is based on research supported by the Austrian Science Fund (FWF) [Grant ID: 10.55776/F85], the Vienna Science and Technology Fund (WWTF) and the City of Vienna [Grant ID: 10.47379/ICT22060], and SBA Research (SBA-K1 NGC), a COMET Center within the COMET – Competence Centers for Excellent Technologies Programme and funded by BMIMI, BMWET, and the federal state of Vienna. The COMET Programme is managed by FFG.

Ethical Considerations

This work analyzes publicly available Android and iOS application packages using automated static analysis. The study did not involve human participants, user interaction, the collection of personal data, or experiments on deployed systems. All analyses were performed offline on publicly distributed application packages and platform documentation. Because this research did not involve human subjects or personal data, it was not submitted to an institutional ethics review board or a comparable external ethics panel.

Open Science

To foster future research, we publish the PermiOSAn taxonomy along with the analysis code, raw data, and other supplementary information at: https://github.com/SecPriv/PermiOSAn. The applications we collected are available upon request.

AI Use

The authors used generative AI-based tools (ChatGPT, Grammarly, and Gemini) to revise the text, improve flow, and correct typos, grammatical errors, and awkward phrasing. We have manually verified and are responsible for the accuracy, originality, and integrity of the output of all AI-based tools.

References

[1] Mathieu Acthernoene. 2026. react-native-permissions API. https://github.com/zoontek/react-native-permissions.

[2] Mathieu Acthernoene. 2026. react-native-permissions API Release 2.0.0. https://github.com/zoontek/react-native-permissions/releases/tag/2.0.0.

[3] Adjust. 2026. SDK Documentation. https://dev.adjust.com/en/sdk.

[4] Mir Masood Ali, David G. Balash, Monica Kodwani, Chris Kanich, and Adam J. Aviv. 2024. Honesty is the Best Policy: On the Accuracy of Apple Privacy Labels Compared to Apps' Privacy Policies. *Proc. of the 23rd Privacy Enhancing Technologies Symposium (PETS)* (2024). doi:10.56553/popets-2024-0111

[5] Ali Alkinoon, Trung Cuong Dang, Ahod Alghuried, Abdulaziz Alghamdi, Soohyeon Choi, Manar Mohaisen, An Wang, Saeed Salem, and David Mohaisen. 2025. A Comprehensive Analysis of Evolving Permission Usage in Android Apps: Trends, Threats, and Ecosystem Insights. *Journal of Cybersecurity and Privacy* 5, 3 (Aug. 2025). doi:10.3390/jcp5030058

[6] Iman M. Almomani and Aala Al Khayer. 2020. A Comprehensive Analysis of the Android Permissions System. *IEEE Access* 2020, 8 (Nov. 2020). doi:10.1109/ACCESS.2020.3041432

[7] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. 2019. PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play. In *Proc. of the 28th USENIX Security Symposium (USENIX Security)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity19/presentation/andow

[8] Apache. 2025. Apache Cordova. https://cordova.apache.org/.

[9] Apple. 2012. About iOS 6. https://support.apple.com/en-us/102995. Archived: https://archive.org/details/do-i-need-to-request-entitlement-d...-apple-developer-forums.

[10] Apple. 2014. iOS Security (February 2014). https://web.archive.org/web/20140226213513/http://images.apple.com/iphone/business/docs/iOS_Security_Feb14.pdf.

[11] Apple. 2014. iOS Security (October 2014). https://web.archive.org/web/20140226213513/https://www.apple.com/mx/privacy/docs/iOS_Security_Guide_Oct_2014.pdf.

[12] Apple. 2023. About the orange and green indicators in your iPhone status bar. https://web.archive.org/web/20250710111833/https://support.apple.com/en-us/108331.

[13] Apple. 2023. If an app would like to connect to devices on your local network. https://support.apple.com/en-us/102229.

[14] Apple. 2024. About iOS 14 Updates. https://web.archive.org/web/20250830111905/https://support.apple.com/en-us/118390.

[15] Apple. 2024. About privacy and Location Services in iOS, iPadOS, and watchOS. https://support.apple.com/en-us/102515.

[16] Apple. 2024. If an app would like to use Bluetooth on your device. https://support.apple.com/en-us/102267.

[17] Apple. 2025. Apple Developer Forum. https://developer.apple.com/forums/thread/735356?answerId=761456022#761456022.

[18] Apple. 2025. Entitlements. http://archive.today/2025.03.20-220255/https://developer.apple.com/documentation/bundleresources/entitlements.

[19] Apple. 2025. NWPathMonitor. http://archive.today/2026.09.03-094047/https://developer.apple.com/documentation/network/nwpathmonitor.

[20] Apple. 2025. Requesting access to protected resources. http://archive.today/2025.05.15-071134/https://developer.apple.com/documentation/uikit/requesting-access-to-protected-resources.

[21] Apple. 2026. Accessory Notifications. https://developer.apple.com/documentation/accessorynotifications.

[22] Apple. 2026. App Extensions. http://archive.today/2026.09.03-090947/https://developer.apple.com/documentation/technologyoverviews/app-extensions.

[23] Apple. 2026. App Tracking Transparency. https://developer.apple.com/documentation/apptrackingtransparency.

[24] Apple. 2026. APS Environment. http://archive.today/2026.09.03-091419/https://developer.apple.com/documentation/bundleresources/entitlements/aps-environment.

[25] Apple. 2026. Complying with Encryption Export Regulations. http://archive.today/2026.09.03-094032/https://developer.apple.com/documentation/Security/complying-with-encryption-export-regulations.

[26] Apple. 2026. Creating a custom keyboard. https://developer.apple.com/documentation/uikit/creating-a-custom-keyboard.

[27] Apple. 2026. iOS & iPadOS Release Notes. http://archive.today/2026.08.18-011521/https://developer.apple.com/documentation/ios-ipados-release-notes.

[28] Apple. 2026. MPMediaPickerController. http://archive.today/2026.09.03-123818/https://developer.apple.com/documentation/mediaplayer/mpmediapickercontroller.

[29] Apple. 2026. Protected resources. http://archive.today/2026.09.03-122702/https://developer.apple.com/documentation/bundleresources/protected-resources.

[30] Apple. 2026. Requesting Authorization to Capture and Save Media. http://archive.today/2026.09.03-092131/https://developer.apple.com/documentation/avfoundation/requesting-authorization-to-capture-and-save-media.

[31] Apple. 2026. UIDocumentPickerViewController. https://developer.apple.com/documentation/uikit/uidocumentpickerviewcontroller.

[32] Apple. 2026. View controllers. https://developer.apple.com/documentation/uikit/view-controllers.

[33] Apple. 2026. We're committed to protecting your data. https://web.archive.org/web/20260221001917/https://www.apple.com/privacy/features/.

[34] Ioannis Arkalakis, Michalis Diamantaris, Serafeim Moustakas, Sotiris Ioannidis, Jason Polakis, and Panagiotis Ilia. 2024. Abandon All Hope Ye Who Enter Here: A Dynamic, Longitudinal Investigation of Android's Data Safety Section. In *Proc. of the 33rd USENIX Security Symposium (USENIX Security)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/arkalakis

[35] David Barrera, H. Güneş Kayacık, Paul van Oorschot, and Anil Somayaji. 2010. A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. In *Proc. of the 17th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM. doi:10.1145/1866307.1866317

[36] Zinaida Benenson, Freya Gassmann, and Lena Reinfelder. 2013. Android and iOS Users' Differences concerning Security and Privacy. In *Proc. of the 32nd ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM. doi:10.1145/2468356.2468502

[37] Kerstin Bongard-Blanchy, Jean-Louis Sterckx, Arianna Rossi, Verena Distler, Salvador Rivas, and Vincent Koenig. 2022. An (Un)Necessary Evil - Users' (Un)Certainty about Smartphone App Permissions and Implications for Privacy Engineering. In *Proc. of the 7th IEEE Symposium on Security & Privacy (S&P)*. IEEE. doi:10.1109/EuroSPW55150.2022.00023

[38] Bram Bonné, Sai Teja Peddinti, Igor Bilogrevic, and Nina Taft. 2017. Exploring decision making with Android's runtime permission dialogs using in-context surveys. In *Proc. of the 13th Symposium On Usable Privacy and Security (SOUPS)*. USENIX Association. https://www.usenix.org/conference/soups2017/technical-sessions/presentation/bonne

[39] Paolo Calciati, Konstantin Kuznetsov, Alessandra Gorla, and Andreas Zeller. 2020. Automatically Granted Permissions in Android apps: An Empirical Study on their Prevalence and on the Potential Threats for Privacy. In *Proc. of the 17th International Conference on Mining Software Repositories (MSR)*. IEEE/ACM. doi:10.1145/3379597.3387469

[40] European Commission. 2025. Digital Markets Act. https://web.archive.org/web/20250809172858/https://digital-markets-act.ec.europa.eu/index_en.

[41] Gael Cooper. 2025. Apple Confirms Switch to iOS 26 Naming Convention: Here's Why That Matters. https://web.archive.org/web/20250819050105/https://www.cnet.com/tech/services-and-software/apple-confirms-switch-to-ios-26-naming-convention-heres-why-that-matters/

[42] Adam Dev. 2026. How to solve missing compliance status in TestFlight and invites not coming to testers. https://web.archive.org/web/20240919105808/https://medium.com/@iamCoder/how-to-solve-missing-compliance-status-in-testflight-and-invites-not-coming-to-testers-4cbbe3c4ed12.

[43] The European Parliament and the Council of the European Union. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union, L 119/1. https://eur-lex.europa.eu/eli/reg/2016/679/oj

[44] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. Android Permissions Demystified. In *Proc. of the 18th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM. doi:10.1145/2046707.2046779

[45] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012. Android Permissions: User Attention, Comprehension, and Behavior. In *Proc. of the 8th Symposium On Usable Privacy and Security (SOUPS)*. USENIX Association. doi:10.1145/2335356.2335360

[46] Google. 2009. Security and Permissions. https://web.archive.org/web/20090213154007/http://developer.android.com/guide/topics/security/security.html.

[47] Google. 2023. Photo Picker Everywhere. https://web.archive.org/web/20260720175309/https://android-developers.googleblog.com/2023/04/photo-picker-everywhere.html

[48] Google. 2025. Permissions on Android. https://web.archive.org/web/20250817231042/https://developer.android.com/guide/topics/permissions/overview.

[49] Google. 2025. Android 13 features and changes list. https://web.archive.org/web/20250803222233/https://developer.android.com/about/versions/12/summary.

[50] Google. 2025. Android 14 features and changes list. https://web.archive.org/web/20250813/https://developer.android.com/about/versions/14/summary.

[51] Google. 2025. Android 7.0 Behavior Changes. https://web.archive.org/web/20250901141310/https://developer.android.com/about/versions/nougat/android-7.0-changes.

[52] Google. 2025. Android 8.0 Behavior Changes. https://web.archive.org/web/20250817073302/https://developer.android.com/about/versions/oreo/android-8.0-changes.

[53] Google. 2025. Android Manifest Permissions. https://web.archive.org/web/20250730191413/https://developer.android.com/reference/android/Manifest.permission.

[54] Google. 2025. Android Open Source Project. https://source.android.com/.

[55] Google. 2025. Android Permission Groups. https://web.archive.org/web/20250715235813/https://developer.android.com/reference/android/Manifest.permission_group.

[56] Google. 2025. Android Restricted Permissions. https://web.archive.org/web/20250711001353/https://support.google.com/googleplay/android-developer/answer/16324062.

[57] Google. 2025. Android Versions. https://developer.android.com/about/versions/.

[58] Google. 2025. Behavior changes: all apps. https://web.archive.org/web/20250821231937/https://developer.android.com/about/versions/pie/android-9.0-changes-all.

[59] Google. 2025. Camera Intents | Android Media. https://web.archive.org/web/20250929083703/https://developer.android.com/media/camera/camera-intents.

[60] Google. 2025. Flutter. https://flutter.dev/.

[61] Google. 2025. Meet Google Play's target API level requirement. web.archive.org/web/2/https://developer.android.com/google/play/requirements/target-sdk.

[62] Google. 2025. Privacy changes in Android 10. https://web.archive.org/web/20250818190352/https://developer.android.com/about/versions/10/privacy/changes.

[63] Google. 2025. Privacy in Android 11 . https://web.archive.org/web/20250706132955/https://developer.android.com/about/versions/11/privacy.

[64] Google. 2025. Request runtime permissions. https://web.archive.org/web/20250830084706/https://developer.android.com/training/permissions/requesting.

[65] Google. 2025. Required actions to comply with our Photo & Video Permissions policy by January 22, 2025. https://support.google.com/googleplay/android-developer/answer/15800983?hl=en

[66] Google. 2025. Sensitive scope verification. https://web.archive.org/web/20250720004927/https://developers.google.com/identity/protocols/oauth2/production-readiness/sensitive-scope-verification.

[67] Google. 2025. Update on Plans for Privacy Sandbox Technologies. https://web.archive.org/web/20251211172125/https://privacysandbox.google.com/blog/update-on-plans-for-privacy-sandbox-technologies.

[68] Google. 2026. Android Compatibility program overview. https://web.archive.org/web/20260818213612/https://source.android.com/docs/compatibility/overview.

[69] Google. 2026. Create a sync adapter. https://web.archive.org/web/20260606083823/https://developer.android.com/training/sync-adapters/creating-sync-adapter.

[70] Google. 2026. Foreground service types. https://web.archive.org/web/20250830111731/https://developer.android.com/develop/background-work/services/fgs/service-types.

[71] Google. 2026. Foreground service typForeground service types are required es. https://web.archive.org/web/20260606104843/https://developer.android.com/about/versions/14/changes/fgs-types-required.

[72] Google. 2026. Mainline. https://web.archive.org/web/20260901231854/https://source.android.com/docs/core/ota/modular-system.

[73] Google. 2026. Required actions to comply with our Photo & Video Permissions policy by January 22, 2025. https://web.archive.org/web/20260619024440/https://support.google.com/googleplay/android-developer/answer/15800983.

[74] Google. 2026. Your Android privacy on your terms. https://web.archive.org/web/20260227110325/https://www.android.com/safety/privacy/.

[75] Daniel Greene and Katie Shilton. 2018. Platform privacies: Governance, collaboration, and the different meanings of "privacy" in iOS and Android development. *New Media & Society* 20, 4 (2018), 1640–1657. doi:10.1177/1461444817702397

[76] Jin Han, Qiang Yan, Debin Gao, Jianying Zhou, and Robert H. Deng. 2013. Comparing Mobile Privacy Protection through Cross-Platform Applications. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*.

[77] Hannah J. Hutton and David A. Ellis. 2023. Exploring User Motivations Behind iOS App Tracking Transparency Decisions. In *Proc. of the 2023 ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3544548.3580654

[78] Paul Jaccard. 1912. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytologist* 11, 2 (Feb. 1912), 37–50. doi:10.1111/j.1469-8137.1912.tb05611.x

[79] JAMF. 2021. An Analysis of iOS App Permissions. https://resources.jamf.com/documents/white-papers/an-analysis-of-ios-app-permissions.pdf

[80] JetBrains. 2025. Kotlin Multiplatform. https://kotlinlang.org/docs/multiplatform.html.

[81] Ryan Johnson, Zhaohui Wang, Corey Gagnon, and Angelos Stavrou. 2012. Analysis of Android Applications' Permissions. In *2012 IEEE Sixth International Conference on Software Security and Reliability Companion (SERE-C)*. doi:10.1109/SERE-C.2012.44

[82] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. 2012. A Conundrum of Permissions: Installing Applications on an Android Smartphone. In *Financial Cryptography and Data Security (FC)*. doi:10.1007/978-3-642-34638-5_6

[83] Rishabh Khandelwal, Asmit Nayak, Paul Chung, and Kassem Fawaz. 2023. Comparing Privacy Labels of Applications in Android and iOS. In *Proc. of the 22nd Workshop on Privacy in the Electronic Society (WPES) (WPES)*. ACM. doi:10.1145/3603216.3624967

[84] Simon Koch, Benjamin Altpeter, and Martin Johns. 2023. The OK Is Not Enough: A Large Scale Study of Consent Dialogs in Smartphone Applications. In *Proc. of the 32nd USENIX Security Symposium (USENIX Security)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity23/presentation/koch

[85] Konrad Kollnig, Anastasia Shuba, Reuben Binns, Max Van Kleek, and Nigel Shadbolt. 2022. Are iPhones Really Better for Privacy? A Comparative Study of iOS and Android Apps. *Proc. of the 22nd Privacy Enhancing Technologies Symposium (PETS)* 2022, 2 (Jan. 2022). doi:10.2478/popets-2022-0033

[86] Jonathan Levin. 2016. **OS Internals: MacOS and IOS Internals, Volume III: Security & Insecurity*. Technologeeks Press.

[87] Rui Li, Wenrui Diao, Zhou Li, Jianqi Du, and Shanqing Guo. 2021. Android Custom Permissions Demystified: From Privilege Escalation to Design Shortcomings. In *Proc. of the 42nd IEEE Symposium on Security & Privacy (S&P)*. IEEE. doi:10.1109/SP40001.2021.00070

[88] Joshua Long. 2016. The Evolution of iOS Security and Privacy Features. https://web.archive.org/web/20250813090729/https://www.intego.com/mac-security-blog/the-evolution-of-ios-security-and-privacy-features/.

[89] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochem Med (Zagreb)* 22, 3 (2012). doi:10.11613/BM.2012.031

[90] Microsoft. 2025. .Net MAUI. https://dotnet.microsoft.com/en-us/apps/maui.

[91] Kirit Modi. 2016. Info.plist - Requesting Permission Privacy Settings in iOS 10. http://archive.today/2026.09.07-132037/https://iosdevcenters.blogspot.com/2016/09/infoplist-privacy-settings-in-ios-10.html.

[92] Reham Mohamed, Arjun Arunasalam, Habiba Farrukh, Jason Tong, Antonio Bianchi, and Z. Berkay Celik. 2024. ATTention Please! An Investigation of the App Tracking Transparency Permission. In *Proc. of the 33rd USENIX Security Symposium (USENIX Security)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/mohamed

[93] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. 2021. Share First, Ask Later (or Never?) Studying Violations of GDPR's Explicit Consent in Android Apps. In *Proc. of the 30th USENIX Security Symposium (USENIX Security)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity21/presentation/nguyen

[94] Trung Tin Nguyen, Michael Backes, and Ben Stock. 2022. Freely Given Consent? Studying Consent Notice of Third-Party Tracking and Its Violations of GDPR in Android Apps. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS) (CCS '22)*. ACM. doi:10.1145/3548606.3560564

[95] Trung Tin Nguyen, Michael Backes, and Ben Stock. 2022. Freely Given Consent? Studying Consent Notice of Third-Party Tracking and Its Violations of GDPR in Android Apps. In *Proc. of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM. doi:10.1145/3548606.3560564

[96] Elleen Pan, Jingjing Ren, Martina Lindorfer, Christo Wilson, and David Choffnes. 2018. Panoptispy: Characterizing Audio and Video Exfiltration from Android Applications. *Proc. of the 18th Privacy Enhancing Technologies Symposium (PETS)* 2018, 4 (2018), 33–50. doi:10.1515/popets-2018-0030

[97] Anthony Peruma, Jeffrey Palmerino, and Daniel E. Krutz. 2018. Investigating User Perception and Comprehension of Android Permission Models. In *Proc. of the 15th International Conference on Mining Software Repositories (MSR)*. IEEE/ACM. doi:10.1145/3197231.3197246

[98] Meta Platforms. 2025. React Native. https://reactnative.dev/.

[99] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. 2014. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *Proc. of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM. doi:10.1145/2660267.2660287

[100] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permission Systems. In *Proc. of the 28th USENIX Security Symposium (USENIX Security)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity19/presentation/reardon

[101] Lena Reinfelder, Zinaida Benenson, and Freya Gassmann. 2014. Differences between Android and iPhone Users in Their Security and Privacy Awareness. In *Trust, Privacy, and Security in Digital Business*. Springer International Publishing, 156–167. doi:10.1007/978-3-319-09770-1_14

[102] Jingjing Ren, Martina Lindorfer, Daniel J. Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. 2018. Bug Fixes, Improvements, ... and Privacy Leaks - A Longitudinal Study of PII Leaks Across Android App Versions. In *Proc. of the 25th Network and Distributed System Security Symposium (NDSS)*. Internet Society (ISOC). doi:10.14722/ndss.2018.23143

[103] Rene Ritchie. 2018. iPhone OS 2.0 review. https://web.archive.org/web/20250808053814/https://www.imore.com/iphone-os-2-review.

[104] David Rodriguez, Akshath Jain, Jose M. Del Alamo, and Norman Sadeh. 2023. Comparing Privacy Label Disclosures of Apps Published in both the App Store and Google Play Stores. In *Proc. of the 8th IEEE European Symposium on Security & Privacy Workshops (EuroS&PW)*. IEEE. doi:10.1109/EuroSPW59978.2023.00022

[105] Neetu Sardana and Arpita Jadhav Bhatt. 2022. CiAFP: Category-based Classification of iOS Apps by Mining Frequent Permissions. In *Proc. of the 2022 Fourteenth International Conference on Contemporary Computing (IC3)*. ACM. doi:10.1145/3549206.3549211

[106] David Schmidt, Alexander Ponticello, Magdalena Steinböck, Katharina Krombholz, and Martina Lindorfer. 2025. Analyzing the iOS Local Network Permission from a Technical and User Perspective. In *Proc. of the 46th IEEE Symposium on Security & Privacy (S&P)*. IEEE. doi:10.1109/SP61157.2025.00045

[107] Bingyu Shen, Lili Wei, Chengcheng Xiang, Yudong Wu, Mingyao Shen, Yuanyuan Zhou, and Xinxin Jin. 2021. Can Systems Explain Permissions Better? Understanding Users' Misperceptions under Smartphone Runtime Permission Model. In *Proc. of the 30th USENIX Security Symposium (USENIX Security)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity21/presentation/shen-bingyu

[108] Signal. 2022. Removing SMS support from Signal Android (soon). https://web.archive.org/web/20260611121404/https://signal.org/blog/sms-removal-android.

[109] Signal. 2026. Commit 3aa54c9. https://github.com/signalapp/Signal-Android/commit/3aa54c99822accc7102226aa0e7f3dee43393be0.

[110] Signal. 2026. Commit aa33fd4. https://github.com/signalapp/Signal-Android/commit/aa33fd44b83263cda70fa4e9f3ce307c604e8719.

[111] Magdalena Steinböck, Jakob Bleier, Mikka Rainer, Tobias Urban, Christine Utz, and Martina Lindorfer. 2024. Comparing Apples to Androids: Discovery, Retrieval, and Matching of iOS and Android Apps for Cross-Platform Analyses. In *Proc. of the 21st International Conference on Mining Software Repositories (MSR) (MSR '24)*. IEEE/ACM. doi:10.1145/3643991.3644896

[112] Mohammad Tahaei, Ruba Abu-Salma, and Awais Rashid. 2023. Stuck in the Permissions With You: Developer & End-User Perspectives on App Permissions & Their Privacy Ramifications. In *Proc. of the 2023 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM. doi:10.1145/3544548.3581060

[113] Güliz Seray Tuncay. 2024. Android Permissions: Evolution, Attacks, and Best Practices. *IEEE Security & Privacy* 22 (Nov. 2024). doi:10.1109/MSEC.2024.3461629

[114] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. 2012. Permission evolution in the Android ecosystem. In *Proc. of the 28th Annual Computer Security Applications Conference (ACSAC)*. ACM. doi:10.1145/2420950.2420956

[115] Jianmao Xiao, Shizhan Chen, Qiang He, Zhiyong Feng, and Xiao Xue. 2020. An Android application risk evaluation framework based on minimum permission set identification. *The Journal of Systems and Software* 163 (Feb. 2020). doi:10.1016/j.jss.2020.110533

[116] Shikun Zhang, Yuanyuan Feng, Yaxing Yao, Lorrie Faith Cranor, and Norman Sadeh. 2022. How Usable Are iOS App Privacy Labels? *Proc. of the 22nd Privacy Enhancing Technologies Symposium (PETS)* 2 (Jan. 2022). doi:10.56553/popets-2022-0106

A Permission Evolution on Android and iOS

Here, we describe the key developments and milestones in Android and iOS permission systems over the years as visualized in Figure 1 in Section 2 and how we identified them. Our analysis sought to highlight how these OSs began to protect user data and progressively enabled users to control which data and resources apps can access. This historical background of how permissions developed on Android and iOS helps to understand both the fundamental differences and commonalities between their permission systems.

To gather insights into the evolution of permission systems on both ecosystems, we used the official documentations of permissions and changelogs of Android and iOS versions where available. Android's documentation provides changelogs and details about all released versions, including new features and changes in permissions [57]. In cases where the changelogs did not provide sufficient insights, we used the Android source code [54] as a supplementary source of information. Apple's documentation of protected resources [29] and entitlements [18] provides the iOS version that introduced the respective permission. iOS release notes are available on Apple's support homepage [27]. Unfortunately, Apple does not provide any information about iPhone OS 1–2 (the previous name for iOS) and iOS 3–5. For these versions, we had to rely on unofficial sources [88].

A.1 Android

In 2008, the first commercial Android device, the HTC Dream, was released, running Android 1.0 (API 1). This version already featured the concept of permissions, whose usage apps needed to declare in order to use protected resources, such as access to the camera [46]. The permissions were granted at install time, and the only way to revoke them was to uninstall the app.

Runtime Permissions. Over the next versions, the number of permissions steadily increased. But it was not until 2015, with the introduction of Android 6 (API 23), that the permission system saw a fundamental change. Apps now have to request permissions to access sensitive data at runtime, and users can choose to allow or deny the request. As of 2026 (Android 16, API 36), some permissions, such as Internet access, are still granted at install time and cannot be revoked. From version Androd 6 on, data about Wi-Fi access points visible to the device were considered location information, and accessing them required an additional permission. In later

versions, additional restrictions were added to better protect users and prevent the abuse of permissions.

Scoped Access. In 2016 (Android 7.0, API 24), an API called "scoped directory access" was introduced to let apps access pictures through item picker dialogs, discouraging the use of `READ_EXTERNAL_STORAGE` [51]. Android 8.0 (API 26) severely limited background access to geolocation data but did not introduce a separate permission for background location access until Android 10 (API 29) [52].

Restrictions & Auto-Revoking Permissions. Android 9 (released 2018, API 28) further restricted background access to data, including sensors. Unlike prior changes, which typically granted exceptions for apps compiled for older Android versions, these restrictions applied to all apps running on Android 9 and newer [58]. A similar change was introduced in Android 10 (released 2019, API 29): Apps developed for a version before Android 6.0, which introduced runtime permissions, would still have all permissions granted at install time. Starting with Android 6.0, users could deselect permissions during the app's first use.

Android 10 introduced a "tristate" location permission: Users could choose to let an app access the location "all the time", "only while the app is in use", or deny the request [62]. In 2020 (Android 11, API 30), some runtime permissions were extended to allow access to functions or resources "only this time", revoking the permission when the app was closed. Permissions would also reset when the system detected that an app had not been used for a while (currently 90 days), revoking permissions that users had already granted. Android 10 also started to enforce "scoped storage", a mechanism to further limit which files apps can access [63]. Since Android 13 (released 2022, API 33), access to the advertising ID, a resettable tracking string, requires a permission that is automatically granted at install time. This version also introduced more granular permissions for access to video, photo, or audio files. Another new permission allows apps to distinguish access to Wi-Fi scan results for networking and tracking purposes [49].

Enforcing Minimum Versions. Starting with Android 14 (released 2023, API 34), old apps with a `targetSdkVersion` below 23 cannot be installed on the system anymore, and Android 15 (release 2025, API 36) further increased this threshold to 24. This prevents exceptions in the permission system from being exploited by malicious apps [50]. The latest Android version at the time of writing (early 2026), Android 16 (released 2025, API 36), did not bring relevant changes, but from August 31, 2025, the Google Play Store requires new apps and app updates to have a `targetSdkVersion` of at least 35. While the `minSdkVersion` can be lower for backward compatibility, the higher requirement makes it more difficult for apps released in the Play Store to exploit compatibility on devices running a modern Android version [61].

A.2 iOS

In 2007, Apple released the iPhone along with iPhone OS 1, which offered only pre-installed apps and a web browser (Safari) but no support for third-party apps [88].

Introduction of Location Services. In iOS 2, released in 2008, Apple first introduced the App Store, allowing third-party developers to bring their own apps onto the iPhone. With this new feature, they also added location services for third-party apps, including a prompt to ask users for permission during runtime [88, 103].

App-Level Permission Controls. With the release of iOS 4 in 2010, Apple added app-level controls for location services to the Settings app, where users could now enable and disable the service for specific apps individually [88]. They further added an icon shown at the top of the screen to indicate that the location services were currently in use [88]. In iOS 6 (released in 2012), Apple added controls to the Settings app for seven different permissions (Location, Contacts, Calendars, Reminders, Photos, Bluetooth Sharing, Accounts (Twitter and Facebook)) [9], which marked the first major update in user privacy controls and the beginning of the modern user-centric privacy and security management on iOS. In February 2014, Apple published a white paper on iOS security, detailing for the first time the security features from hardware to software and establishing iOS as a security-focused platform [10]. The extended version of this document, updated in October 2014 [11], also contains a section dedicated to "Privacy Controls", incorporating the finer controls introduced with the release of iOS 8 a month prior. This update added controls for Motion & Fitness and HomeKit data and replaced the simple toggles of allow/disallow in the privacy control panel of the Settings app with more granular options, e.g., "while in use" or "always" for location services. Apple further added controls for camera and microphone access, raising the number of user-controllable permissions to eleven.

Rationales. Another new feature for permission controls was introduced with the release of iOS 10 in 2016: Apple added mandatory rationales (also called "usage descriptions" or "purpose strings") for all protected resources. Developers now had to specify why they requested a certain permission from the user by adding the rationales to the app's `Info.plist` file [91] via XML key-value pairs. This marks Apple's first attempt to bring more transparency into the collection of sensitive data on iOS.

One-Time & Restricted-Access Permissions. With iOS 13 (released in 2019), Apple introduced one-time permissions, i.e., the option "allow once" for location services, meaning that the user can allow access to the location only once, and after a restart the app has to ask for permission again [15]. The update also restricted the use of Bluetooth by requiring a permission prompt for everything except audio playback [16]. In iOS 14 (2020), Apple added a permission to protect the local network and restricted access to the photo library, where the user can now select individual photos or videos that they want to make accessible to an app [13]. This update also improved the indicator of active camera and microphone use, which is now a green dot at the top of the screen if the camera is in use and an orange dot if the microphone is in use [12]. Another new feature was App Tracking Transparency (ATT), which allows users to select whether an app may track them or not [14]. With iOS 17 (released in 2023), Apple improved the permission controls for the calendar and reminders through finer-grained options, i.e., "none", "add only" or "full access." With the release of iOS 18 in 2024, Apple improved the privacy controls for the Contacts permission, allowing users to select which contacts they want to share with an app instead of only allowing full or no access.

B Alternative Approaches to Mapping Permission Categories Across Platforms

Beyond the method presented in Section 3, we investigated other options to map app permissions between iOS and Android. While we ultimately opted against these, we report them here, as they could be useful to future work in cross-platform app analysis.

Apple's Permission Grouping. One approach to map app permissions across platforms could be to leverage the grouping Apple provides for all permissions across all its platforms (e.g., "Photos", "Location", or "Sensors") [29] and apply it to other platforms. However, this grouping is often not directly related to the scope of a permission, but rather a general technical category (e.g., NFC entitlements in Networking). There are also inconsistent category names for entitlements and protected resources, which would create an ambiguity in the mapping, e.g., "Wireless Interfaces" and "NFC". Finally, some groups refer to proprietary Apple services, e.g., "FinanceKit" handles on-device financial data. In conclusion, Apple's grouping is not generalizable and therefore cannot serve as an out-of-the-box solution for a cross-platform analysis.

Android Permission Groups. Android provides 18 "permission groups" [55], which logically cluster related permissions and are primarily used to reduce the number of system dialogs. For example, reading and writing calendar entries are separate permissions, but the system treats them as a single group. Users are only shown one dialog for both permissions and if one is granted, the other permissions in the same group are also granted. These groups are too fine-grained and do not contain all related permissions to adequately capture all common permission use cases. This makes them unsuitable for use in a universal methodology for mapping permissions across mobile platforms.

Permission Handling in Cross-Platform Frameworks. Another potential approach to permission mapping could be offered by cross-platform app development frameworks. In order to allow for efficient development and deployment of apps on both Android and iOS, these frameworks require a heuristic approach to access comparable resources, including permissions, on both platforms. We analyzed five popular development frameworks (Cordova [8], Flutter [60], React Native [98], Kotlin Multiplatform [80], and .Net MAUI [90]) and found that they only provide a subset of available permissions or require developers to manually configure permissions. React Native and .NET MAUI provide handling of a subset of permissions, but (1) React Native requires requesting permissions separately on both platforms, which requires knowledge about the respective counterpart on the other OS, (2) .NET MAUI only provides a subset of available permissions for both platforms, requiring manual implementation for unsupported permissions, and (3) and both frameworks require developers to manually configure permissions in the app manifests. Flutter, Cordova, and Kotlin Multiplatform do not natively support permission abstraction but leave this to additional libraries and plugins. Thus, cross-platform development frameworks do not provide a broad basis to create a mapping methodology but highlight the practical need for such a mapping to simplify cross-platform development processes.

Using Large Language Models to Derive Categories. We also explored whether Large Language Models (LLMs) could be used to map Android and iOS permissions into cross-platform categories. We ran some experiments with different prompts and LLMs (e.g., ChatGPT) but observed that LLM-created mappings lacked an understanding of permission scopes and the underlying mechanisms of the Android and iOS ecosystems, making them unsuitable for our purposes. Thus, we opted for a manual approach that leverages human reasoning and extensive domain knowledge.

C Extended Analysis Results

Supporting our analysis in Section 4.2, Table 6 shows the evolution of the average number of requested XPPCs on each platform over time. Figure 5 plots the number of requested XPPCs for all Android–iOS app pairs for 2023 and 2025. Differences in the number of XPPCs between app pairs occur quite often and are, on average, between 3 to 4. 2,925 (98.7%) app pairs exhibited at least one difference in XPPC adoption across platforms over the years. This indicates platform-specific behavior within the same apps. In 2023, 75 (2.5%) app pairs showed identical XPPCs across both platforms, increasing to 79 (2.7%) in 2024 and to 87 (2.9%) in 2025.

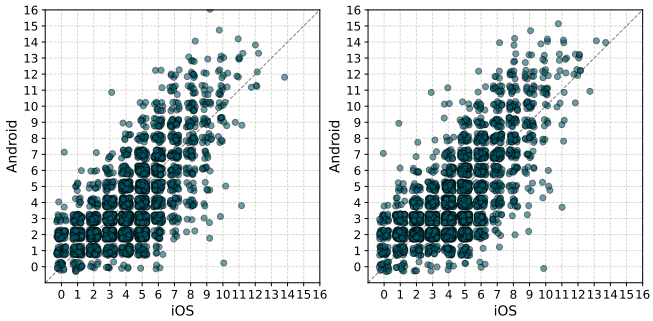

Figure 5: Requested XPPCs per app pair in 2023 (left) and 2025 (right). Each dot represents the number of permissions an app requires on each OS. A dot on the diagonal means that an app requests the same number, dots above or below represent Android and iOS apps requesting more permissions, respectively. Apps requested an increasing number on Android, on iOS the number remained stable.

Table 6: Average number of requested XPPCs per App Store category for Android and iOS from 2023 to 2025. On Android, the average number of permissions rose for most App Store categories, whereas on iOS it declined in many categories.

	2023		2024		2025	
Social Networking	7.45	6.26	7.59	6.39	7.64	6.30
Lifestyle	6.80	6.28	7.12	6.49	7.24	6.63
Business	6.60	5.79	7.04	5.94	7.13	5.99
Travel	6.52	5.75	6.65	6.11	6.75	6.04
Finance	6.45	6.05	7.05	6.23	7.40	6.40
Navigation	6.39	5.58	6.58	5.94	6.72	5.83
Utilities	6.32	5.86	6.71	6.24	6.86	6.30
Shopping	5.80	5.53	5.89	5.66	6.01	5.62
Food & Drink	5.51	5.26	5.54	5.37	5.77	5.51
Productivity	5.42	5.42	5.61	5.57	5.76	5.54
Health & Fitness	4.92	5.34	5.50	5.65	5.78	5.58
Photo & Video	4.86	5.06	5.08	5.18	5.15	5.21
Music	4.78	5.15	4.99	5.29	5.20	5.36
Medical	4.64	4.84	4.84	4.84	4.92	4.84
Magazines & Newspapers	4.50	4.50	5.00	4.50	5.00	4.50
Reference	4.00	3.73	4.15	3.88	4.27	3.85
News	3.56	4.29	4.00	4.37	4.17	4.41
Weather	3.56	3.89	3.52	3.93	3.59	3.89
Entertainment	3.49	4.72	3.73	4.84	3.77	4.93
Books	3.40	3.60	3.55	3.70	3.95	3.55
Education	3.28	3.43	3.37	3.57	3.57	3.56
Graphics & Design	3.23	3.95	3.36	4.18	3.41	4.23
Sports	3.05	3.83	3.22	3.91	3.26	3.81
Strategy	2.96	3.57	3.09	3.54	3.13	3.55
Role Playing	2.93	3.74	3.01	3.73	3.11	3.68
Board	2.84	3.22	2.90	3.35	2.96	3.20
Casino	2.76	4.03	2.97	4.13	3.03	4.16
Word	2.64	2.94	2.58	2.88	2.55	2.82
Adventure	2.58	2.98	2.58	3.00	2.57	2.92
Games	2.56	2.67	2.94	2.61	3.06	2.56
Action	2.54	3.14	2.72	3.12	2.84	3.10
Developer Tools	2.50	4.50	3.00	4.50	3.00	4.50
Card	2.50	3.76	2.50	3.69	2.57	3.69
Simulation	2.29	3.32	2.49	3.34	2.52	3.35
Trivia	2.29	2.58	2.25	2.50	2.42	2.75
Puzzle	2.26	3.10	2.32	3.07	2.32	3.03
Casual	2.16	2.72	2.35	2.73	2.41	2.69
Racing	2.12	2.84	2.40	2.84	2.44	2.63
Family	1.90	2.98	2.14	3.00	2.33	3.06

D Intents and Single-Platform Permissions

D.1 Intents

For our analysis in Section 4.3, we collected all relevant Android Intents from the Intent documentation as well as the MediaStore documentation. We excluded Intents that required an existing reference to a resource, such as `android.intent.action.EDIT`. Table 7 shows our mapping for Intent constant names that could be assigned a single XPPC. For iOS, Table 8 shows relevant interfaces. Some of them, such as `UIImagePickerController`, can be used to ask the user to take a picture, in which case the `NSCameraUsageDescription` is required. The same interface can also be used to pick images; then no additional permission is required.

Table 7: Android Intents in our app pair dataset that correspond to an XPPC.

Android Intent	XPPC
android.media.action.IMAGE_CAPTURE	Camera
android.media.action.IMAGE_CAPTURE_SECURE	
android.media.action.STILL_IMAGE_CAMERA	
android.media.action.STILL_IMAGE_CAMERA_SECURE	
android.media.action.VIDEO_CAPTURE	
android.provider.action.MOTION_PHOTO_CAPTURE	
android.provider.action.MOTION_PHOTO_CAPTURE_SECURE	
android.intent.action.CREATE_DOCUMENT	Media
android.intent.action.OPEN_DOCUMENT	
android.intent.action.OPEN_DOCUMENT_TREE	
android.provider.action.PICK_IMAGES	
android.intent.action.DIAL	Telephony Services

Table 8: Intent-like interfaces on iOS in our app pair dataset that correspond to an XPPC.

iOS Interfaces	XPPC
EKEventEditViewController	Camera
CNContactPickerViewController	Contacts
PHPickerViewController	Media

D.2 Android-only Permissions

Table 9 shows the Android-only permissions of the apps in our dataset (see Section 4.4), i.e., permissions with no counterpart on iOS.

Table 9: Android-only permissions in our app dataset of n = 2,964 cross-platform app pairs. Long permission prefixes (`android.permission.`, `com.android.*`) have been omitted for readability.

Permission	#Apps	%	Permission	#Apps	%
ACCESS_NETWORK_STATE	2960	99.87	EXPAND_STATUS_BAR	27	0.91
INTERNET	2960	99.87	FOREGROUND_SERVICE_SYSTEM_EXEMPTED	22	0.74
BIND_JOB_SERVICE	2890	97.5	BIND_QUICK_SETTINGS_TILE	44	1.48
WAKE_LOCK	2880	97.17	BIND_NOTIFICATION_LISTENER_SERVICE	43	1.45
FOREGROUND_SERVICE	2670	90.08	KILL_BACKGROUND_PROCESSES	39	1.32
VIBRATE	2098	70.78	SET_WALLPAPER	37	1.25
RECEIVE_BOOT_COMPLETED	1517	51.18	UNINSTALL_SHORTCUT	33	1.11
READ_PHONE_STATE	701	23.65	HIDE_OVERLAY_WINDOWS	33	1.11
FOREGROUND_SERVICE_DATA_SYNC	618	20.85	EXPAND_STATUS_BAR	27	0.91
USE_FINGERPRINT	487	16.43	FOREGROUND_SERVICE_SYSTEM_EXEMPTED	22	0.74
GET_ACCOUNTS	331	11.17	RUN_USER_INITIATED_JOBS	18	0.61
BIND_REMOTEVIEWS	262	8.84	FOREGROUND_SERVICE_REMOTE_MESSAGING	17	0.57
USE_CREDENTIALS	247	8.33	BIND_ACCESSIBILITY_SERVICE	16	0.54
REORDER_TASKS	229	7.73	REQUEST_DELETE_PACKAGES	14	0.47
AUTHENTICATE_ACCOUNTS	158	5.33	BIND_INPUT_METHOD	12	0.4
MANAGE_ACCOUNTS	156	5.26	BIND_WALLPAPER	11	0.37
FLASHLIGHT	134	4.52	READ_USER_DICTIONARY	9	0.3
INSTALL_SHORTCUT	134	4.52	DETECT_SCREEN_RECORDING	9	0.3
GET_TASKS	132	4.45	RESTART_PACKAGES	7	0.24
DETECT_SCREEN_CAPTURE	130	4.39	WRITE_USER_DICTIONARY	6	0.2
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	112	3.78	SUBSCRIBED_FEEDS_WRITE	6	0.2
USE_FULL_SCREEN_INTENT	99	3.34	SUBSCRIBED_FEEDS_READ	6	0.2
WRITE_SYNC_SETTINGS	98	3.31	BIND_COMPANION_DEVICE_SERVICE	6	0.2
ACCESS_NOTIFICATION_POLICY	95	3.21	READ_HISTORY_BOOKMARKS	4	0.13
READ_SYNC_SETTINGS	84	2.83	WRITE_PROFILE	4	0.13
FOREGROUND_SERVICE_CONNECTED_DEVICE	83	2.8	USE_SIP	4	0.13
FOREGROUND_SERVICE_SPECIAL_USE	81	2.73	REQUEST_PASSWORD_COMPLEXITY	4	0.13
READ_BASIC_PHONE_STATE	81	2.73	BIND_CONTROLS	4	0.13
QUERY_ALL_PACKAGES	70	2.36	WRITE_HISTORY_BOOKMARKS	3	0.1
BROADCAST_STICKY	62	2.09	BIND_VOICE_INTERACTION	3	0.1
DISABLE_KEYGUARD	54	1.82	BIND_TEXT_SERVICE	3	0.1
GET_PACKAGE_SIZE	50	1.69	SET_WALLPAPER_HINTS	2	0.07
SET_ALARM	49	1.65	ADD_VOICEMAIL	2	0.07
READ_SYNC_STATS	49	1.65	WRITE_SMS	2	0.07
READ_PROFILE	45	1.52	TRANSMIT_IR	2	0.07
USE_EXACT_ALARM	45	1.52	BIND_DREAM_SERVICE	2	0.07
BIND_QUICK_SETTINGS_TILE	44	1.48	WRITE_SOCIAL_STREAM	1	0.03
BIND_NOTIFICATION_LISTENER_SERVICE	43	1.45	READ_SOCIAL_STREAM	1	0.03
KILL_BACKGROUND_PROCESSES	39	1.32	BIND_MUSIC_RECOGNITION_SERVICE	1	0.03
SET_WALLPAPER	37	1.25	BIND_TV_INPUT	1	0.03
UNINSTALL_SHORTCUT	33	1.11	BIND_HOTWORD_DETECTION_SERVICE	1	0.03
HIDE_OVERLAY_WINDOWS	33	1.11	BIND_PRINT_SERVICE	1	0.03
EXPAND_STATUS_BAR	27	0.91	BIND_MIDI_DEVICE_SERVICE	1	0.03
FOREGROUND_SERVICE_SYSTEM_EXEMPTED	22	0.74			

D.3 iOS-only Permissions

Table 10 shows the iOS-only permissions of the apps in our dataset (see Section 4.5), i.e., permissions with no counterpart on Android.

Table 10: iOS-only permissions in our app dataset of n = 2,964 cross-platform app pairs. Long permission prefixes (`com.apple.*`, `com.apple.developer.*`) have been omitted for readability.

Permission	# Apps	%	Permission	# Apps	%
ITSAppUsesNonExemptEncryption	2,300	77.6	previous-application-identifiers	7	0.24
NSUserTrackingUsageDescription	2,215	74.73	pushkit.unrestricted-voip	7	0.24
associated-domains	1,929	65.08	web-browser	6	0.2
applesignin	1,453	49.02	video-subscriber-single-sign-on	6	0.2
security.application-groups	1,233	41.6	ClassKit-environment	5	0.17
UIRequiresPersistentWiFi	1,217	41.06	NSDocumentsFolderUsageDescription	5	0.17
NSCalendarsUsageDescription	1,102	37.18	NSFileProviderDomainUsageDescription	5	0.17
keychain-access-groups	1,081	36.47	coremedia.allow-alternate-video-decoder-selection	5	0.17
NSLocationAlwaysUsageDescription	959	32.35	NSNearbyInteractionUsageDescription	5	0.17
game-center	659	22.23	weatherkit	4	0.13
NSBluetoothPeripheralUsageDescription	551	18.59	NSGKFriendListUsageDescription	4	0.13
icloud-container-identifiers	410	13.83	carplay-driving-task	4	0.13
ubiquity-kvstore-identifier	347	11.71	carplay-charging	4	0.13
icloud-services	297	10.02	carplay-messaging	4	0.13
icloud-container-environment	294	9.92	homekit.allow-setup-payload	4	0.13
NSSpeechRecognitionUsageDescription	266	8.97	family-controls	4	0.13
in-app-payments	240	8.1	navigation-app	4	0.13
ubiquity-container-identifiers	229	7.73	networking.manage-thread-network-credentials	4	0.13
siri	189	6.38	carplay-communication	3	0.1
NSLocationUsageDescription	166	5.6	carplay-calling	3	0.1
usernotifications.time-sensitive	154	5.2	carplay-parking	3	0.1
NSSiriUsageDescription	97	3.27	browser.app-installation	3	0.1
ITSEncryptionExportComplianceCode	86	2.9	translation-app	3	0.1
networking.HotspotConfiguration	83	2.8	mediasetup	3	0.1
usernotifications.communication	82	2.77	legacyvoip	3	0.1
pass-type-identifiers	78	2.63	carplay-fueling	2	0.07
NSRemindersUsageDescription	78	2.63	contactless-payment-pass-provisioning	2	0.07
devicecheck.appattest-environment	77	2.6	in-app-identity-presentment	2	0.07
default-data-protection	71	2.4	in-app-identity-presentment.merchant-identifiers	2	0.07
kernel.increased-memory-limit	52	1.75	NSIdentityUsageDescription	2	0.07
carplay-audio	46	1.55	usernotifications.filtering	2	0.07
payment-pass-provisioning	44	1.48	journal.allow	2	0.07
external-accessory.wireless-configuration	43	1.45	networking.slicing.trafficcategory	2	0.07
kernel.extended-virtual-addressing	35	1.18	networking.slicing.appcategory	2	0.07
associated-appclip-app-identifiers	34	1.15	private.cloudkit.systemService	1	0.03
NSHomeKitUsageDescription	28	0.94	edu-assessment-mode	1	0.03
device-information.user-assigned-device-name	24	0.81	financekit	1	0.03
usernotifications.critical-alerts	23	0.78	carkey.session	1	0.03
smoot.subscriptionservice	22	0.74	carplay-protocols	1	0.03
group-session	22	0.74	sensitivecontentanalysis.client	1	0.03
passkit.pass-presentation-suppression	21	0.71	storekit.request-data	1	0.03
homekit	20	0.67	NSNearbyInteractionAllowOnceUsageDescription	1	0.03
playable-content	20	0.67	enrollment-sso-capable	1	0.03
user-fonts	17	0.57	NSAppleEventsUsageDescription	1	0.03
networking.HotspotHelper	17	0.57	NSDownloadsFolderUsageDescription	1	0.03
carplay-maps	16	0.54	NSAppleScriptEnabled	1	0.03
inter-app-audio	16	0.54	coremedia.allow-mpeg4streaming	1	0.03
networking.multipath	15	0.51	sustained-execution	1	0.03
avfoundation.multitasking-camera-access	13	0.44	springboard.opensensitiveurl	1	0.03
matter.allow-setup-payload	13	0.44	itunesstored.privacy-acknowledged	1	0.03
shared-with-you	12	0.4	itunesstored.private	1	0.03
NSVideoSubscriberAccountUsageDescription	12	0.4	storekit.cloud-service-exempted-from-tcc-access	1	0.03
private.memory.legacy_footprint	11	0.37	hearing-aid-app	1	0.03
mail-client	10	0.34	pushkit.unrestricted-voip-regulatory	1	0.03
storekit.request-data	8	0.27	push-to-talk	1	0.03
proximity-reader.payment.acceptance	7	0.24	pushkit.unrestricted-voip.ptt	1	0.03
storekit.external-link.account	7	0.24			

E Full PermiOSAn Taxonomy

Table 11: Full list of 21 cross-platform permission categories (XPPCs), mapping 251 Android permissions to 50 iOS permissions. Legend: ● protection level normal (Android), ⚠ protection level dangerous (Android), 🧲 bind permission (Android), 👤 confirmed by the user, 📱 confirmed by the OS, 🔒 confirmed by vendor (Apple, Google), 👥 requires device manager.

iOS Permission(s)	XPPC	Android Permission(s)
📱 com.apple.developer.accessory-setup-discovery-extension	Accessory Setup	● 📱 REQUEST_COMPANION_PROFILE_GLASSES
		● 📱 REQUEST_COMPANION_PROFILE_WATCH
		● 📱 REQUEST_COMPANION_RUN_IN_BACKGROUND
		● 📱 REQUEST_COMPANION_START_FOREGROUND_SERVICES_FROM_BACKGROUND
		● 📱 REQUEST_COMPANION_USE_DATA_IN_BACKGROUND
		● 📱 REQUEST_OBSERVE_COMPANION_DEVICE_PRESENCE
		● 📱 DELIVER_COMPANION_MESSAGES
📱 com.apple.developer.authentication-services.autofill-credential-provider	Authentication	🧲 📱 BIND_CREDENTIAL_PROVIDER_SERVICE
		● 📱 BIND_AUTOFILL_SERVICE
👤 NSFaceIDUsageDescription	Biometrics	● 📱 USE_BIOMETRIC
👤 NSBluetoothAlwaysUsageDescription	Bluetooth	⚠ 👤 BLUETOOTH_ADVERTISE
		⚠ 👤 BLUETOOTH_CONNECT
		⚠ 👤 BLUETOOTH_SCAN
		● 📱 BLUETOOTH
		● 📱 BLUETOOTH_ADMIN
👤 NSCalendarsFullAccessUsageDescription	Calendar	⚠ 👤 READ_CALENDAR
👤 NSCalendarsWriteOnlyAccessUsageDescription		⚠ 👤 WRITE_CALENDAR
👤 NSCameraUsageDescription	Camera	⚠ 👤 CAMERA
		● 📱 FOREGROUND_SERVICE_CAMERA
👤 NSContactsUsageDescription	Contacts	⚠ 👤 READ_CONTACTS
🔒 com.apple.developer.contacts.notes		⚠ 👤 WRITE_CONTACTS
👤 NSLocalNetworkUsageDescription	Local Network	⚠ 👤 NEARBY_WIFI_DEVICES
📱 com.apple.developer.location.push	Location	⚠ 👤 ACCESS_COARSE_LOCATION
👤 NSLocationAlwaysAndWhenInUseUsageDescription		⚠ 👤 ACCESS_FINE_LOCATION
👤 NSLocationWhenInUseUsageDescription		⚠ 🔒 ACCESS_BACKGROUND_LOCATION
👤 NSLocationTemporaryUsageDescriptionDictionary		⚠ 👤 ACCESS_MEDIA_LOCATION
👤 NSWidgetWantsLocation		⚠ 👤 RANGING
👤 NSLocationDefaultAccuracyReduced		⚠ 👤 UWB_RANGING
		● 📱 FOREGROUND_SERVICE_LOCATION
		● 📱 ACCESS_LOCATION_EXTRA_COMMANDS
👤 NSPhotoLibraryAddUsageDescription	Media	⚠ 👤 READ_MEDIA_VISUAL_USER_SELECTED
👤 NSPhotoLibraryUsageDescription		⚠ 🔒 READ_MEDIA_IMAGES
👤 NSAppleMusicUsageDescription		⚠ 🔒 READ_MEDIA_VIDEO
		⚠ 👤 READ_EXTERNAL_STORAGE
		⚠ 👤 WRITE_EXTERNAL_STORAGE
		⚠ 👤 READ_MEDIA_AUDIO
		● 📱 MODIFY_AUDIO_SETTINGS
		● 📱 FOREGROUND_SERVICE_MEDIA_PLAYBACK
		● 📱 FOREGROUND_SERVICE_MEDIA_PROCESSING
		● 📱 FOREGROUND_SERVICE_MEDIA_PROJECTION
📱 com.apple.developer.messages.critical-messaging	Messaging	⚠ 🔒 READ_SMS
📱 com.apple.developer.messaging-app		⚠ 🔒 RECEIVE_MMS
📱 com.apple.developer.carrier-messaging-app		⚠ 🔒 RECEIVE_SMS
		⚠ 🔒 RECEIVE_WAP_PUSH
		⚠ 🔒 SEND_SMS
		🧲 📱 BIND_CARRIER_MESSAGING_CLIENT_SERVICE
👤 NSMicrophoneUsageDescription	Microphone	⚠ 👤 RECORD_AUDIO
		⚠ 👤 FOREGROUND_SERVICE_MICROPHONE
👤 NSMotionUsageDescription	Motion	⚠ 👤 ACTIVITY_RECOGNITION
👤 NSFallDetectionUsageDescription		● 📱 HIGH_SAMPLING_RATE_SENSORS
🔒 com.apple.developer.networking.multicast	Multicast	● 📱 CHANGE_WIFI_MULTICAST_STATE
📱 com.apple.developer.networking.networkextension	Networking	● 📱 CHANGE_NETWORK_STATE
		● 📱 CHANGE_WIFI_STATE
👤 NFCReaderUsageDescription	NFC	● 📱 NFC
📱 com.apple.developer.nfc.readersession.formats		● 📱 NFC_PREFERRED_PAYMENT_INFO
🔒 com.apple.developer.nfc.hce		● 📱 NFC_TRANSACTION_EVENT
🔒 com.apple.developer.nfc.hce.iso7816.select-identifier-prefixes		🧲 📱 BIND_NFC_SERVICE
🔒 com.apple.developer.nfc.hce.default-contactless-app		
👤 NSUserNotificationsUsageDescription	Notification	⚠ 👤 POST_NOTIFICATIONS
📱 aps-environment		
📱 com.apple.developer.calling-app	Telephony Services	⚠ 🔒 READ_CALL_LOG
📱 com.apple.developer.dialing-app		⚠ 👤 ACCEPT_HANDOVER
🔒 com.apple.CommCenter.fine-grained		⚠ 👤 ADD_VOICEMAIL
📱 com.apple.developer.networking.carrier-constrained.appcategory		⚠ 👤 CALL_PHONE
📱 com.apple.developer.networking.carrier-constrained.app-optimized		⚠ 🔒 WRITE_CALL_LOG
		⚠ 👤 READ_PHONE_NUMBERS
		⚠ 👤 ANSWER_PHONE_CALLS
		● 📱 CALL_COMPANION_APP
		● 📱 MANAGE_OWN_CALLS
		● 📱 FOREGROUND_SERVICE_PHONE_CALL
		⚠ 🔒 PROCESS_OUTGOING_CALLS
		🧲 📱 BIND_SCREENING_SERVICE
		🧲 📱 BIND_TELECOM_CONNECTION_SERVICE
		🧲 📱 BIND_INCALL_SERVICE

Table 11: Full list of 21 cross-platform permission categories, mapping 251 Android permissions to 50 iOS permissions. (cont.)

iOS	XPPC	Android
		BIND_CALL_REDIRECTION_SERVICE
com.apple.developer.networking.vpn.api	VPN	BIND_VPN_SERVICE
com.apple.developer.networking.wifi-info	Wi-Fi State	ACCESS_WIFI_STATE
com.apple.developer.automated-device-enrollment.add-devices	Device Management	BIND_DEVICE_ADMIN
		MANAGE_DEVICE_POLICY_ACCESSIBILITY
		MANAGE_DEVICE_POLICY_ACCOUNT_MANAGEMENT
		MANAGE_DEVICE_POLICY_ACROSS_USERS
		MANAGE_DEVICE_POLICY_ACROSS_USERS_FULL
		MANAGE_DEVICE_POLICY_ACROSS_USERS_SECURITY_CRITICAL
		MANAGE_DEVICE_POLICY_AIRPLANE_MODE
		MANAGE_DEVICE_POLICY_APP_FUNCTIONS
		MANAGE_DEVICE_POLICY_APP_RESTRICTIONS
		MANAGE_DEVICE_POLICY_APP_USER_DATA
		MANAGE_DEVICE_POLICY_APPS_CONTROL
		MANAGE_DEVICE_POLICY_ASSIST_CONTENT
		MANAGE_DEVICE_POLICY_AUDIO_OUTPUT
		MANAGE_DEVICE_POLICY_AUTOFILL
		MANAGE_DEVICE_POLICY_BACKUP_SERVICE
		MANAGE_DEVICE_POLICY_BLOCK_UNINSTALL
		MANAGE_DEVICE_POLICY_BLUETOOTH
		MANAGE_DEVICE_POLICY_BUGREPORT
		MANAGE_DEVICE_POLICY_CALLS
		MANAGE_DEVICE_POLICY_CAMERA
		MANAGE_DEVICE_POLICY_CAMERA_TOGGLE
		MANAGE_DEVICE_POLICY_CERTIFICATES
		MANAGE_DEVICE_POLICY_COMMON_CRITERIA_MODE
		MANAGE_DEVICE_POLICY_CONTENT_PROTECTION
		MANAGE_DEVICE_POLICY_DEBUGGING_FEATURES
		MANAGE_DEVICE_POLICY_DEFAULT_SMS
		MANAGE_DEVICE_POLICY_DEVICE_IDENTIFIERS
		MANAGE_DEVICE_POLICY_DISPLAY
		MANAGE_DEVICE_POLICY_FACTORY_RESET
		MANAGE_DEVICE_POLICY_FUN
		MANAGE_DEVICE_POLICY_INPUT_METHODS
		MANAGE_DEVICE_POLICY_INSTALL_UNKNOWN_SOURCES
		MANAGE_DEVICE_POLICY_KEEP_UNINSTALLED_PACKAGES
		MANAGE_DEVICE_POLICY_KEYGUARD
		MANAGE_DEVICE_POLICY_LOCALE
		MANAGE_DEVICE_POLICY_LOCATION
		MANAGE_DEVICE_POLICY_LOCK
		MANAGE_DEVICE_POLICY_LOCK_CREDENTIALS
		MANAGE_DEVICE_POLICY_LOCK_TASK
		MANAGE_DEVICE_POLICY_MANAGED_SUBSCRIPTIONS
		MANAGE_DEVICE_POLICY_METERED_DATA
		MANAGE_DEVICE_POLICY_MICROPHONE
		MANAGE_DEVICE_POLICY_MICROPHONE_TOGGLE
		MANAGE_DEVICE_POLICY_MOBILE_NETWORK
		MANAGE_DEVICE_POLICY_MODIFY_USERS
		MANAGE_DEVICE_POLICY_MTE
		MANAGE_DEVICE_POLICY_NEARBY_COMMUNICATION
		MANAGE_DEVICE_POLICY_NETWORK_LOGGING
		MANAGE_DEVICE_POLICY_ORGANIZATION_IDENTITY
		MANAGE_DEVICE_POLICY_OVERRIDE_APN
		MANAGE_DEVICE_POLICY_PACKAGE_STATE
		MANAGE_DEVICE_POLICY_PHYSICAL_MEDIA
		MANAGE_DEVICE_POLICY_PRINTING
		MANAGE_DEVICE_POLICY_PRIVATE_DNS
		MANAGE_DEVICE_POLICY_PROFILE_INTERACTION
		MANAGE_DEVICE_POLICY_PROFILES
		MANAGE_DEVICE_POLICY_PROXY
		MANAGE_DEVICE_POLICY_QUERY_SYSTEM_UPDATES
		MANAGE_DEVICE_POLICY_RESET_PASSWORD
		MANAGE_DEVICE_POLICY_RESTRICT_PRIVATE_DNS
		MANAGE_DEVICE_POLICY_RUN_IN_BACKGROUND
		MANAGE_DEVICE_POLICY_RUNTIME_PERMISSIONS
		MANAGE_DEVICE_POLICY_SAFE_BOOT
		MANAGE_DEVICE_POLICY_SCREEN_CAPTURE
		MANAGE_DEVICE_POLICY_SCREEN_CONTENT
		MANAGE_DEVICE_POLICY_SECURITY_LOGGING
		MANAGE_DEVICE_POLICY_SETTINGS
		MANAGE_DEVICE_POLICY_SMS
		MANAGE_DEVICE_POLICY_STATUS_BAR
		MANAGE_DEVICE_POLICY_SUPPORT_MESSAGE
		MANAGE_DEVICE_POLICY_SUSPEND_PERSONAL_APPS
		MANAGE_DEVICE_POLICY_SYSTEM_APPS
		MANAGE_DEVICE_POLICY_SYSTEM_DIALOGS
		MANAGE_DEVICE_POLICY_SYSTEM_UPDATES
		MANAGE_DEVICE_POLICY_THREAD_NETWORK
		MANAGE_DEVICE_POLICY_TIME
		MANAGE_DEVICE_POLICY_USB_DATA_SIGNALLING
		MANAGE_DEVICE_POLICY_USB_FILE_TRANSFER

Table 11: Full list of 21 cross-platform permission categories, mapping 251 Android permissions to 50 iOS permissions. (cont.)

iOS	XPPC	Android
		MANAGE_DEVICE_POLICY_USERS
		MANAGE_DEVICE_POLICY_VPN
		MANAGE_DEVICE_POLICY_WALLPAPER
		MANAGE_DEVICE_POLICY_WIFI
		MANAGE_DEVICE_POLICY_WINDOWS
		MANAGE_DEVICE_POLICY_WIPE_DATA
		MANAGE_DEVICE_POLICY_APP_FUNCTIONS
NSHealthClinicalHealthRecordsShareUsageDescription	Health	BODY_SENSORS
NSHealthShareUsageDescription		BODY_SENSORS_BACKGROUND
NSHealthUpdateUsageDescription		FOREGROUND_SERVICE_HEALTH
NSHealthRequiredReadAuthorizationTypeIdentifiers		READ_ACTIVE_CALORIES_BURNED
com.apple.developer.healthkit		READ_ACTIVITY_INTENSITY
com.apple.developer.healthkit.access		READ_BASAL_BODY_TEMPERATURE
com.apple.developer.healthkit.background-delivery		READ_BASAL_METABOLIC_RATE
com.apple.developer.healthkit.recalibrate-estimates		READ_BLOOD_GLUCOSE
com.apple.developer.health.fall-detection		READ_BLOOD_PRESSURE
		READ_BODY_FAT
		READ_BODY_TEMPERATURE
		READ_BODY_WATER_MASS
		READ_BONE_MASS
		READ_CERVICAL_MUCUS
		READ_DISTANCE
		READ_ELEVATION_GAINED
		READ_EXERCISE
		READ_EXERCISE_ROUTES
		READ_FLOORS_CLIMBED
		READ_HEALTH_DATA_HISTORY
		READ_HEALTH_DATA_IN_BACKGROUND
		READ_HEART_RATE
		READ_HEART_RATE_VARIABILITY
		READ_HEIGHT
		READ_HYDRATION
		READ_INTERMENSTRUAL_BLEEDING
		READ_LEAN_BODY_MASS
		READ_MEDICAL_DATA_ALLERGIES_INTOLERANCES
		READ_MEDICAL_DATA_CONDITIONS
		READ_MEDICAL_DATA_LABORATORY_RESULTS
		READ_MEDICAL_DATA_MEDICATIONS
		READ_MEDICAL_DATA_PERSONAL_DETAILS
		READ_MEDICAL_DATA_PRACTITIONER_DETAILS
		READ_MEDICAL_DATA_PREGNANCY
		READ_MEDICAL_DATA_PROCEDURES
		READ_MEDICAL_DATA_SOCIAL_HISTORY
		READ_MEDICAL_DATA_VACCINES
		READ_MEDICAL_DATA_VISITS
		READ_MEDICAL_DATA_VITAL_SIGNS
		READ_MENSTRUATION
		READ_MINDFULNESS
		READ_NUTRITION
		READ_OVULATION_TEST
		READ_OXYGEN_SATURATION
		READ_PLANNED_EXERCISE
		READ_POWER
		READ_RESPIRATORY_RATE
		READ_RESTING_HEART_RATE
		READ_SEXUAL_ACTIVITY
		READ_SKIN_TEMPERATURE
		READ_SLEEP
		READ_SPEED
		READ_STEPS
		READ_TOTAL_CALORIES_BURNED
		READ_VO2_MAX
		READ_WEIGHT
		READ_WHEELCHAIR_PUSHES
		WRITE_ACTIVE_CALORIES_BURNED
		WRITE_ACTIVITY_INTENSITY
		WRITE_BASAL_BODY_TEMPERATURE
		WRITE_BASAL_METABOLIC_RATE
		WRITE_BLOOD_GLUCOSE
		WRITE_BLOOD_PRESSURE
		WRITE_BODY_FAT
		WRITE_BODY_TEMPERATURE
		WRITE_BODY_WATER_MASS
		WRITE_BONE_MASS
		WRITE_CERVICAL_MUCUS
		WRITE_DISTANCE
		WRITE_ELEVATION_GAINED
		WRITE_EXERCISE
		WRITE_EXERCISE_ROUTE
		WRITE_FLOORS_CLIMBED
		WRITE_HEART_RATE

Table 11: Full list of 21 cross-platform permission categories, mapping 251 Android permissions to 50 iOS permissions. (cont.)

iOS	XPPC	Android
		WRITE_HEART_RATE_VARIABILITY
		WRITE_HEIGHT
		WRITE_HYDRATION
		WRITE_INTERMENSTRUAL_BLEEDING
		WRITE_LEAN_BODY_MASS
		WRITE_MEDICAL_DATA
		WRITE_MENSTRUATION
		WRITE_MINDFULNESS
		WRITE_NUTRITION
		WRITE_OVULATION_TEST
		WRITE_OXYGEN_SATURATION
		WRITE_PLANNED_EXERCISE
		WRITE_POWER
		WRITE_RESPIRATORY_RATE
		WRITE_RESTING_HEART_RATE
		WRITE_SEXUAL_ACTIVITY
		WRITE_SKIN_TEMPERATURE
		WRITE_SLEEP
		WRITE_SPEED
		WRITE_STEPS
		WRITE_TOTAL_CALORIES_BURNED
		WRITE_VO2_MAX
		WRITE_WEIGHT
		WRITE_WHEELCHAIR_PUSHES