The Medium is the Message: How Secure Messaging Apps Leak Sensitive Data to Push Notification Services

Nikita Samarin,[1,2] Alex Sanchez,[1] Trinity Chung,[1] Akshay Dan Bhavish Juleemun,[1] Conor Gilsenan,[1] Nick Merrill,[1] Joel Reardon,[3] and Serge Egelman[1,2]
{nsamarin,alexso,trinityc,adbjuleemun,conorgilsenan,ffff,egelman}@berkeley.edu
joel.reardon@ucalgary.ca
[1]University of California, Berkeley; [2]International Computer Science Institute (ICSI); [3]University of Calgary

ABSTRACT

Like most modern software, secure messaging apps rely on third-party components to implement important app functionality. Although this practice reduces engineering costs, it also introduces the risk of inadvertent privacy breaches due to misconfiguration errors or incomplete documentation. Our research investigated secure messaging apps' usage of Google's Firebase Cloud Messaging (FCM) service to send push notifications to Android devices. We analyzed 21 popular secure messaging apps from the Google Play Store to determine what personal information these apps leak in the payload of push notifications sent via FCM. Of these apps, 11 leaked metadata, including user identifiers (10 apps), sender or recipient names (7 apps), and phone numbers (2 apps), while 4 apps leaked the actual message content. Furthermore, none of the data we observed being leaked to FCM was specifically disclosed in those apps' privacy disclosures. We also found several apps employing strategies to mitigate this privacy leakage to FCM, with varying levels of success. Of the strategies we identified, none appeared to be common, shared, or well-supported. We argue that this is fundamentally an economics problem: incentives need to be correctly aligned to motivate platforms and SDK providers to make their systems secure and private by default.

KEYWORDS

privacy, security, mobile, push notifications, FCM

1 INTRODUCTION

> *"She speaks, yet she says nothing."*
> —William Shakespeare, *Romeo and Juliet*

Modern economies rely on the specialization of labor [74]. Software engineering is no different: modern software relies on myriad third-party components to fulfill tasks so that developers do not need to waste time rebuilding specific functions from scratch [28]. This type of "code reuse" is a recommended practice and transcends many branches of engineering (e.g., car manufacturers do not manufacture every component that goes into their cars, instead relying on components from third-party suppliers). Software development kits (SDKs) facilitate code reuse during software development and offer many benefits for developers. They provide well-trodden paths:

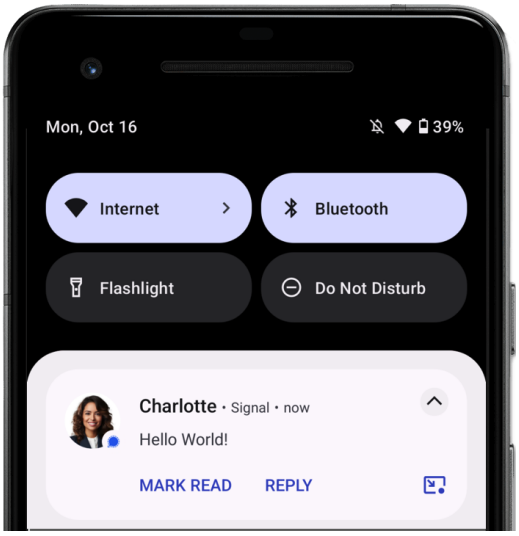

Figure 1: An illustration of an Android push notification.

documented workflows for developers to follow so that these developers can consistently provide common functionality. Ultimately, SDKs reduce engineering costs when used responsibly.

Yet, recent research has demonstrated that many software privacy issues (i.e., the inappropriate disclosure of sensitive user information) are due to developers' misuse of third-party services [4, 65]. That is, privacy breaches often occur due to developers not correctly configuring SDKs, not reading SDK documentation, or SDKs behaving in undocumented ways, often unbeknownst to developers. This is especially concerning when the third-party SDK may transmit highly sensitive user data to third parties and the SDK is ubiquitous across many software supply chains.

Heightened public concerns around the monitoring of online communications have significantly influenced consumer behavior in the past decade. A 2014 PEW survey found that 70% of Americans are concerned about government surveillance and 80% about surveillance by corporations [53]. In response to these concerns, more and more consumers have begun using secure messaging apps to protect their communications based on the promises of privacy made by these apps. Hundreds of millions of users now use apps like Signal or Telegram, believing these apps to protect their privacy. These applications are entrusted with a vast array of confidential user data, from personal conversations to potentially-sensitive multimedia content, thereby placing a significant emphasis on their ability to make good on their promises of privacy and security.

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–16

https://doi.org/XXXXXXX.XXXXXXX

The misuse of third-party SDKs within secure messaging apps may pose a heightened risk to users because those SDKs may leak sensitive information to third parties. In particular, app developers use third-party SDKs to implement *push notifications*, which display important information to the user, including messages from other app users (Figure 1). Because push notification SDKs are generally provided by third parties (as opposed to app developers), incorrect usage may leak sensitive information to those third parties. For example, an app that provides "end-to-end" encrypted messaging may not actually provide end-to-end encryption if message payloads are not encrypted before being sent to third-party push notification APIs. To make matters worse, misuse of these SDKs may also contribute to the misrepresentation of security and privacy assurances to consumers as articulated in various disclosures, including privacy policies, terms of service, and marketing materials.

The combined risk of sensitive information leakage and misrepresentation of privacy promises creates serious ramifications for users of secure messaging platforms. Oppressive regimes or other adversaries may use court orders to compel companies involved in the delivery infrastructure of push notifications to reveal the contents of communications sent and received by human-rights workers, political dissidents, journalists, etc. Worse, when this does happen, both the developers of the apps and the users who are endangered are unlikely to be aware that their communications are being intercepted. This threat model is not just theoretical. Crucially, since we performed our analysis, U.S. Senator Ron Wyden published an open letter that confirms that government agencies do, in fact, collect user information by demanding push notification records from Google and other push notification providers through the use of legal processes [100]. Our work is highly prescient, as it provides new insights into an emergent threat model.

To study the extent to which the delivery infrastructure may access sensitive user information, we examined the use of Google's *Firebase Cloud Messaging* (FCM) to deliver push notifications to *secure messaging apps* on Android devices. Google provides FCM as a free service, and therefore, it is one of the most commonly used third-party SDKs to deliver Android push notifications. Moreover, the majority of other push services, including OneSignal [58], Pusher [63], and AirShip [3] internally rely on Google's FCM to deliver notifications to Android devices, making the usage of FCM practically unavoidable for developers who wish to provide push notification support in their Android apps. (On Apple's iOS, third-party push notification APIs are similarly built on top of Apple's push notification service [59].) We focus on secure messaging apps because these apps (1) market their abilities to keep message data "private" or "secure" and (2) make heavy use of push notifications to notify users of incoming messages and their contents (and therefore, when not implemented correctly, may run the risk of leaking message contents and metadata to the push notification service).

Prior work has investigated the potential security risks that push notifications may pose, including by push notification-based malware [41, 48] and botnets [41, 47]. To our knowledge, no work has focused on the privacy risks of push notification services used by secure messaging apps. Therefore, we performed a study to examine whether the push notification records potentially stored without end-to-end encryption by the delivery infrastructure may misrepresent or compromise the privacy protections of secure messaging and expose users to legal risks. Thus, we posed the following research questions:

- **RQ1:** What personal data do secure messaging apps for Android send via Google's Firebase Cloud Message (FCM)?
- **RQ2:** What mitigation strategies do app developers use to protect personal information from being disclosed to Google's FCM?
- **RQ3:** Do the observed data-sharing behaviors align with the privacy assurances apps make in their public disclosures?

To answer these questions, we performed static and dynamic analysis on a corpus of 21 secure messaging apps. We used dynamic analysis to understand what data these apps sent over the network. When we found that apps displayed data in push notifications, but did not obviously send that data over the network, we used static analysis to understand what mitigation strategies they used to achieve this effect. In contrast, when segments of data displayed in the app *were* verbatim in push notifications, we further examined these messages to assess whether sensitive data was available in plaintext to the delivery infrastructure. Finally, we analyzed apps' privacy policies and other disclosures to identify the privacy claims that apps made to users. By comparing observed behavior from our app analysis to disclosed behavior, we identify undisclosed sharing and potentially-misleading data practices: data that apps imply that they will not disclose, but—intentionally or not—do disclose to the delivery infrastructure through the use of push notifications.

We found that more than half of the apps in our corpus leak *some* personal information to Google via FCM. Furthermore, none of the data we observed being leaked to FCM was specifically disclosed in those apps' privacy disclosures. We also found several apps employing strategies to mitigate this privacy leakage to FCM, with varying levels of success. Of those identified strategies, none appeared to be common, shared, or well-supported. While app developers are ultimately responsible for the behavior of their apps, they are often ill-equipped to evaluate their apps' privacy and security properties in practice. Given that the problems that we observe are pervasive across app developers and stem from the use of third-party components that can be easily used insecurely, we conclude that SDK providers are best positioned to fix these types of issues through both better guidance and privacy-preserving designs and defaults.

In this paper, we contribute the following:

- We demonstrate the widespread sharing of personal information, perhaps inadvertently, with Google through developers' use of push notifications.
- We highlight systemic mismatches between privacy disclosures and observed behaviors in delivering push notifications via FCM.
- We discuss developers' negligence in deploying software that they do not understand and the responsibility that SDK and platform providers share in creating infrastructures that are private/secure by default.

2 BACKGROUND

We provide an overview of push notification services (PNS), specifically Google's Firebase Cloud Messaging (FCM). We describe the threat model we consider in this paper and our overall motivation.

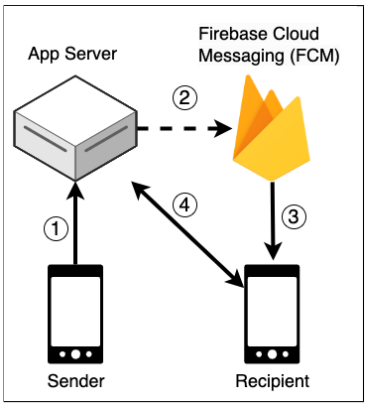

Figure 2: Flow chart of FCM's push notification infrastructure for messaging apps, highlighting the actors involved and the interactions between them: an event occurs that triggers a push notification, e.g., a message from a sender (1) prompts the app server to create and send the message to FCM (2), which then forwards it to the recipient's Android device (3). If needed, the receiving app running on that device may also query additional information from the app server (4).

2.1 Mobile Push Notifications

A push notification is a short message that appears as a pop-up on the desktop browser, mobile lock screen, or in a mobile device's notification center (Figure 1). Push notifications are typically opt-in[1] alerts that display text and rich media, like images or buttons, which enable a user to take a specific action in a timely fashion, even when the app in question is in the background. Applications often use push notifications as a marketing or communication channel, but they can also be used as a security mechanism (e.g., as part of a multi-factor authentication ceremony).

There is a difference between push messages and notifications. "Push" is the technology for sending messages from the server-side component of the app (the "app server") to its client side (the "client app"), even when the user is not actively using the app. Notifications refer to the process of displaying timely information to the user by the app's user interface (UI) [12]. In the context of mobile apps, the application server can send a push message without displaying a notification (i.e., a silent push); an app can also display a notification based on an in-app event without receiving any push messages. For simplicity's sake, we use the term "push notifications" in this paper regardless of whether an actual notification is displayed to the end user (i.e., we refer to messages flowing through a cloud messaging server to a user's device, whereupon the device's operating system routes the messages to the appropriate app).

Although app developers could, in theory, implement their own push notification service, this is usually impractical as it requires the app to continually run as a background service, thereby reducing battery life. Instead, most mobile app developers rely on *operating system push notification services* (OSPNSs), including Firebase Cloud Messaging (FCM) for Android or Apple Push Notification Service (APNS) for iOS devices [8]. FCM and other PNSs facilitate push notifications via an SDK the developer adds to their application. When a user launches the app for the first time, the SDK registers the device with the PNS by generating a *push token* (also known as a *registration token*), which serves as a pseudonymous identifier that tells the push service where to forward the messages. The SDK returns the push token to the client app, which should then be sent and stored in a database on the app server. When the app wants to send a push notification, it looks up the appropriate push token and sends it alongside the message to the PNS, which then forwards the message to the correct device [94]. The push token is tied to the app instance, and therefore, the developer should periodically refresh it, e.g., if the user deletes and reinstalls the app.

In sum, there are three main actors involved in the process of sending push notifications using FCM (see also Figure 2):

App Server sends event-specific messages to FCM (2). For instance, in the context of a messaging app, a sender device may send a message to the app server (1), which then sends a push notification request to FCM (2).

Firebase Cloud Messaging (FCM) is a cloud-based OSPNS that forwards push messages to the appropriate user device using the stored registration token(3), even if the client app is offline or in the background. It also exposes an API to the developer to enable push messaging in their applications.

Android Device runs the OS and the client app. Android uses a system component that is part of Google Play Services to receive push messages sent by FCM, which it then passes to the appropriate app. Optionally, the client app can also query additional information from the app server (4) in response to a received push notification.

The SDKs distributed by FCM and other PNSs not only streamline app development by reducing the amount of code that needs to be written, but in many cases, their use is necessary for performance and efficiency reasons [79]. Developers would also need to request the Android permission for unrestricted battery usage, something a user might not necessarily grant. As such, mobile platform owners only provide official support for their managed OSPNSs: Google for FCM and Apple for ASPNS.[2]

2.2 FCM Alternatives

Given the utility of push notifications, companies have started offering push notification services that compete with Google's FCM. These third-party PNS providers, such as Airship, Pushwoosh, and Taplytics, may offer advantages over FCM, including more features or usable APIs. While it may seem that developers using third-party PNSs can potentially avoid the security and privacy pitfalls of FCM, Lou et al. demonstrated that third-party push providers rely on FCM to deliver messages to Android devices with Google Play Services [52]. The authors identified the dual-platform structure of push notifications. The first service ("host notification platform") abstracts push messaging by providing an API that interfaces with the second service ("transit notification platform"), which provides a stable system-level communication channel to deliver push notifications to user devices. While both FCM and third-party PNSs offer developer-facing APIs for managing push notifications (i.e.,

[1] Android and iOS require user permission before an app can display notifications.

[2] We studied Android because the operating system is open source, allowing us to more easily build instrumentation to monitor app execution.

the host notification platform), only FCM fulfills the role of the transit notification platform and delivers messages internally to Android devices with Google Play Services.

Furthermore, we found statements by several popular third-party PNSs, such as OneSignal [58], Pusher [63], and AirShip [3] that mention their dependence on FCM for sending push notifications to Android devices. For instance, OneSignal states in a blog post that "Google mandates that Android apps distributed through Google Play leverage a single, shared connection provided by FCM" and "OneSignal itself uses the FCM API internally to send messages to Android devices" [58]. Therefore, these third-party PNSs expose users to risks associated with FCM push notifications while potentially introducing their own problematic data collection practices. For instance, Reuters has previously reported that Pushwoosh—a third-party PNS—misrepresented itself as based in the U.S. despite actually being headquartered in Russia [61]. Although Pushwoosh denied the claims [46], the revelation still led the U.S. Army and Centers for Disease Control and Prevention (CDC) to stop using apps containing the Pushwoosh SDK.

Android devices without preinstalled Google Play Services either do not properly support push notifications or use an alternative platform. Most notably, Android devices sold in China do not include Google Play Services, but use another preinstalled service provided by the phone manufacturer, such as Huawei Mobile Services (HMS), to handle push notifications. There are other Android variants outside of China that do not come with Google Play Services preinstalled, such as FireOS, which runs on Amazon devices and uses Amazon Device Messaging (ADM) instead of FCM. These variants constitute a small share of the global Android market [31] and are outside the scope of our analysis.

Other alternatives, such as UnifiedPush [86] or Samsung Push Service [22], rely on apps to receive push notifications in place of Google Play Services. However, we argue that such solutions do not represent equivalent alternatives, as they require users to install an additional app and developers may still use FCM as the push service, unbeknownst to app users. Thus, we specifically focus on data shared with Google's FCM, regardless of the specific third-party service running on top of it. (That is, our instrumentation is agnostic as to whether it captured messages sent natively using FCM or another third-party API built upon it.)

2.3 Threat Model

FCM acts as an intermediary between the server-side and client-side applications and uses push tokens to identify the device where push notifications should be forwarded. While efficient, this architecture poses three significant privacy risks to users [27, 99]:

- **Disclosure.** The contents of a push notification and its metadata may be disclosed to unauthorized entities.
- **Linking.** Push tokens may be linked or attributed to specific users or behaviors.
- **Identification.** Individuals may become identified based on the information linked to their device's push tokens.

The primary threat model that we consider is the use of legal processes to request FCM push tokens linked to a targeted device and stored by the app developer. In the context of secure messaging apps, knowing the pseudonym (i.e., username) of the targeted user may suffice. Even if the app developer does not collect other identifying personal information, they must still store registration tokens to route the push notifications to the user's device through FCM servers. After obtaining the push tokens from the app publisher, law enforcement can request that Google provide all information linked to the given push token, which may include the contents and metadata of the associated push notifications. Combining these pieces of personal information increases the risk of identification.

This threat model is not theoretical. In December 2023, U.S. Senator Ron Wyden published an open letter confirming that government agencies collect user information by demanding push notification records from Google and Apple through legal processes [100]. Since then, journalists found more than 130 search warrants and court orders going back to 2019 (e.g., [20, 87, 88]) in which investigators had demanded that tech companies, notably Wickr and TeleGuard—both advertised as end-to-end encrypted secure messaging apps—turn over push tokens associated with accounts of specific users. In the case of TeleGuard, an FBI agent then asked Google to hand over all information connected to the push token, which Google responded to with account names and IP addresses associated with those accounts [40]. Furthermore, Apple disclosed in its transparency report for the second half of 2022 that it received 70 requests worldwide seeking identifying information about Apple Accounts (formerly known as Apple IDs) associated with 794 push tokens and provided data in response to 54 (77%) requests. Google does not specifically break out government requests for push notification records and, instead, reports these requests in aggregate with other account data requests [9].

We hypothesize that many Android app developers transmit sensitive information via established third-party push notification channels and do not realize that they are not properly securing it. In a departure from "privacy-by-design" principles [16], the official Google Android Developers Blog recommends [69] that developers using Google's service "send as much data as possible in the [push notification] payload" and fetch the remainder of the data from the app server if needed. In the next paragraph of the blog, developers are advised that they "can also encrypt FCM messages end-to-end using libraries like Capillary," thereby indicating that FCM does not encrypt payload data by default (i.e., developers need to rely on additional libraries). There is no other mention of end-to-end encryption in the blog. Thus, questions remain as to whether developers follow this optional guidance.

Google's FCM developer documentation [36] states that "depending on your needs, you may decide to add end-to-end encryption to data messages" and "FCM does not provide an end-to-end solution." No further guidance is given on what information is appropriate to send. In contrast, Apple's documentation for sending notifications [8] instructs developers not to include "customer information or any sensitive data in a notification's payload" and, if they must, "encrypt it before adding it to the payload." Even if the majority of data sent using push notification channels is not personal, there are examples in which it might be, such as some user-generated content in instant messaging apps or sensitive information sent by a banking or a health-tracking app. In these cases, app vendors may be held liable for failing to safeguard or minimize the amount of personal information sent via push notification servers and for failing to disclose this practice in their privacy notices.

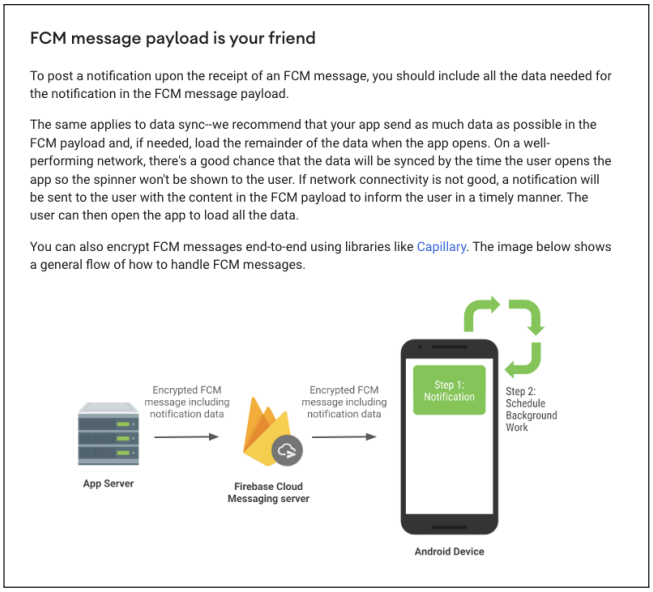

Figure 3: Google's guidance to send as much data as possible via FCM payloads, noting that end-to-end encryption can optionally be used via additional libraries [69]. It is unclear whether the data flows labeled "encrypted" refer to this option or the fact that the transmissions use TLS.

Given FCM's role as an intermediary, we posed the question: do apps leak user information through push notifications to the delivery infrastructure? We investigated this question by performing both mobile app analysis and analysis of privacy disclosures.

3 RELATED WORK

In this section, we provide an overview of related work on the privacy and security risks of push notifications, mobile app analysis, and analysis of privacy-relevant disclosures.

3.1 Risks of Push Notifications

Prior research has demonstrated how attackers can exploit mobile push notifications to spam users with advertisements [50], launch phishing attacks [102], and even issue commands to botnets [2, 41, 47]. Other studies have revealed additional security issues with PNSs that can result in the loss of confidentiality (i.e., user messages get exposed to unauthorized parties) and integrity (i.e., users receive malicious messages from unauthorized parties) [17]. By assuming that the victim installs a malicious app, prior work has demonstrated how attackers can abuse platform-provided OSPNSs, including Google's FCM (formerly known as Google Cloud Messaging or GCM, and Cloud to Device Messaging or C2DM prior to that), to steal sensitive messages or even remotely control the victim's device [48]. Warren et al. described "security" as a key nonfunctional requirement for implementing push notification mechanisms and identified the push-to-sync strategy back in 2014 (which they called "poke-and-pull") as a viable protection strategy for protecting user data from PNSs [93].

As described previously (§ 2.2), push notification architecture can be separated into the host platform that provides the push API and the transit platform that actually delivers the push notification internally. Several studies looked at the security issues of third-party PNS SDKs while excluding system-level transit platforms, such as FCM from Google. One study analyzed 30 different third-party PNS SDKs embedded in 35,173 Android apps and found that 17 SDKs contain vulnerabilities to the confidentiality and integrity of push messages, which an attacker can exploit by running a malicious app on the victim's device [17]. Similarly, Lou et al. performed a security and privacy analysis of the twelve most popular PNSs and compared their behavior in 31,049 apps against information practices disclosed in the privacy policies of those PNSs [52]. They found that out of twelve third-party PNSs, six PNSs collect in-app user behavior and nine collect location information, often without awareness or consent of app users. As the authors focused only on the host platforms, their analysis excluded FCM (and other transit platforms) on the basis of them being a "trustful service provider." We complement this work by focusing instead on the privacy risks of transit notification platforms, in particular, FCM from Google.

In recent years, researchers have analyzed PNSs from the perspective of privacy protection goals that complement the classic "CIA triad" (confidentiality, integrity, and availability), such as unlinkability, transparency, and intervenability [38]. One study, for instance, considered an adversary with the capability to silently sniff packets directed to or from the victim and actively trigger push notification messages to the target's personal device [51]. The authors demonstrated that under these assumptions, an adversary on the same network can deidentify the victim even if they use an online pseudonym. We complement these studies by focusing on FCM privacy risks in the context of secure messaging apps.

3.2 Mobile App Analysis

Numerous studies have also investigated the security and privacy ramifications of mobile apps (e.g., [26, 43, 77, 83]). Most current methods for evaluating mobile app actions depend on static analysis [30, 37, 44, 105], which examines the app's source code without executing it. However, this technique is limited as it can only identify the potential behaviors of a program, not if and to what degree the program exhibits them. For instance, it is generally infeasible to predict the full set of execution branches that a program will take. Alternative methods, such as taint tracking [23], which tracks the flow of data as it propagates through the application, come with their own challenges, including affecting app stability [15].

A newer approach involves adding instrumentation to the Android operating system to monitor apps' access to personal information at runtime [84, 95–97]. This allows researchers to investigate different app behaviors, including app-associated network traffic. Prior solutions to monitoring mobile app transmissions generally involve using proxy software (e.g., Charles Proxy,[3] mitmproxy,[4] etc.) and suffer from serious shortcomings. First, they route all the device traffic through the proxy, without automatically attributing traffic to a specific app running on the device. While some traffic may contain clues (e.g., content and headers that may identify apps,

[3]https://www.charlesproxy.com/
[4]https://mitmproxy.org/

e.g., HTTP `User-Agent` headers), other traffic does not, and attributing traffic to the app is a laborious and uncertain process [64]. Second, proxies often cannot automatically decode various obfuscations, including TLS with certificate pinning. Instead, by capturing traffic from the monitored device's OS, these issues are eliminated. This approach can bypass certificate pinning, extract decryption keys from memory, and map individual sockets to process names, thereby offering precise attribution to specific apps.

3.3 Analysis of Privacy Disclosures

Prior research has focused on understanding apps' and websites' privacy practices by analyzing disclosures made in privacy policies [7, 39, 92, 104, 105]. Some proposed systems, such as POLICHECK [7], MAPS [104] and HPDROID [24], which automate the process of comparing disclosures made in privacy policies about how user data is used, collected, or shared with personal data transmissions observed as a result of performing technical analyses [7, 72, 92, 104, 105]. The literature also proposed systems, such as Polisis [39], PI-Extract [14] and PrivacyFlash [103], which made it possible to transform privacy policies into formats that are more understandable to users or autogenerate policies that reflect actual app behaviors. Linden et al. [49] found that disclosures made in privacy policies improved as a result of GDPR enforcement, but that more improvements would have to be made before they can be considered usable and transparent to users. Other recent studies have also examined the accuracy of disclosures made in privacy policies [6, 57, 68, 92].

Additionally, Google's Play Store requires developers to provide privacy labels [35]. Privacy labels communicate information practices to users in a visually succinct way. For example, apps may list the data types (e.g., names, phone numbers, identifiers) collected and shared with third parties. As with privacy policies, these privacy labels are required by the Google Play Store's terms of service to be thorough and complete [35]. However, Google states in their guidelines that "transferring user data to a 'service provider'" should not be disclosed as data sharing in the app's privacy labels [35], limiting their scope and potential utility. Other studies have also demonstrated the inconsistencies between privacy labels and privacy policies [76], privacy labels in the Google Play Store and Apple App Store for the same apps [66], and practices disclosed in privacy labels and behaviors observed among iOS apps [45, 101].

4 METHODS

Our primary research question concerns how secure messaging apps' usage of FCM impacts user privacy. To answer this question, we identified a set of apps from the Google Play Store and compared the claims made in their privacy disclosure documents with our static and dynamic analysis of those same apps.

The diagram in Figure 2 outlines the main actors and communications involved in push notification usage in secure messaging apps. The messaging app is installed on the phone/device of the sender and the recipient. First, the sender composes their message, and some content gets sent over the network to the app's server (1). Then, the server uses the FCM API to construct the push notification with the required payload. The FCM API sends the notification to Google's FCM server (2), which then forwards it to the recipient device (3) using a long-lived TCP connection initiated by Google Play Services. Finally, the data is parsed and packed into an intent that is then broadcast to the app, which displays the message in the form of a notification. Inadvertent data leakage to Google occurs when the server places user information as plaintext in the push notification payload. Crucially, users and developers are likely unaware that Google may receive and, sometimes, retain[5] message contents and other metadata associated with the push notification.

As highlighted in § 3, numerous prior works evaluate the security and privacy of end-to-end (e2e) encryption and its implementation in secure messaging apps, including many of the ones in our corpus. However, our work is explicitly **not** investigating these claims of e2e encryption. Therefore, we are not interested in recording the traffic sent over a network connection. Rather, our interest is in determining whether implementing push notification functionality in a given app leaks personal message content to parties *other than the app developer*, specifically to Google via FCM. Therefore, we are primarily interested in what data the app's server sends to FCM via network connection. However, because we are out-of-band from this network connection, the best alternative is to record the inbound/outbound traffic on the recipient's device to infer which data may have been sent from the server to FCM. If the sender's plaintext message content is present in the push notification sent to the recipient's device from FCM, then it is clear that the app server did leak the user's message content to FCM. However, if the push notification sent to the recipient's device does not contain the sender's plaintext message, then it may be likely that the app server did **not** leak the user's message content to FCM.[6] For apps that fall into this category, we additionally want to understand the techniques they leverage to avoid leaking user message content and metadata to FCM.

4.1 App Selection

We selected messaging apps that made claims about the privacy of users' messages (herein, "secure messaging apps"). For example, Telegram's homepage promotes its app as "private" and states that "Telegram messages are heavily encrypted" [78]. Similarly, Signal's homepage encourages people to "speak freely" because the Signal app has a "focus on privacy" [71]. Signal publicly writes about what data their app collects and the fact that—in response to a legal subpoena requesting a range of user information—Signal is only able to provide "timestamps for when each account was created and the date that each account last connected to the Signal service" [70]. WhatsApp also explicitly markets the privacy benefits of their app and states, "[y]our privacy is our priority. With end-to-end encryption, you can be sure that your personal messages stay between you and who you send them to" [80, 81]. Because secure messaging apps make these claims about the privacy of users' messages, many users utilize these apps in sensitive contexts. For example, Telegram, Signal, and WhatsApp, three of the apps we analyzed, are frequently used by protesters worldwide [73, 89]. The apps in our data set, a subset of all secure messaging apps, are widely used and encompass over 2.8 billion users and 6.1 billion installs.

[5] E.g., FCM servers retain messages by default when the recipient device is offline.
[6] If the app server has access to the sender's plaintext message, then it is always possible that it is leaked to third-parties in ways that are not externally detectable, since traffic between the app server and these third parties is not observable.

Material Representations. The selection of messaging apps based on their privacy claims is not only a prudent approach for users prioritizing the confidentiality of their communications, but also a legally-grounded strategy, reflecting the enforceable nature of such assertions. When companies publicly assert their services' privacy and security features, these claims become material representations that can significantly influence consumer choices. Importantly, material misrepresentations are actionable under consumer protection laws. For instance, under the FTC Act[7] (and various state consumer protection laws), businesses in the U.S. are prohibited from materially misrepresenting their practices to consumers. The Federal Trade Commission (FTC) and state attorneys general actively monitor and pursue companies that fail to uphold their privacy promises (regardless of whether they are made in privacy policies [18] or marketing materials [19]). This enforcement protects consumers and reinforces the message that privacy and security assertions are material representations that have legal consequences and can affect consumer choices.

One such notable case is that of Zoom, in which the company faced a regulatory enforcement action for erroneously claiming to offer end-to-end encryption in its marketing materials, a feature it did not fully provide at the time [25]. This incident underscores the seriousness with which authorities treat misrepresentations in the digital privacy domain, highlighting the risks companies face when they do not accurately describe their data protection measures. Thus, evaluating messaging apps based on their stated privacy features is not only a measure of their utility in sensitive contexts, but also an assessment of their compliance with legal standards for truthfulness in advertising, ensuring that users can rely on the integrity of these claims.

Selection Procedure. We aimed to create a corpus of secure messaging apps that made privacy claims to users, such that it included widely-used apps and was of a tractable size to perform our analyses. To create this corpus, we first had to identify a set of the most popular secure messaging apps in the Google Play Store. We focused on apps in the `Communication` category in the Google Play Store, which included a broad range of messaging apps, including email clients, mobile browsers, and SMS apps. Within this category, we used open-source tooling[8] to identify apps whose descriptions included one or more keywords related to online messaging[9] and explicitly excluded keywords related to non-messaging apps.[10]

To establish this list of keywords, we manually reviewed the descriptions of apps in the `Communication` category and iteratively added keywords to our inclusion and exclusion lists until we manually determined that the resulting set of apps included secure messaging apps that do not fall back onto SMS. Then, we excluded any app whose description did not include the terms "privacy" or "security." Finally, we only selected apps with more than a million installations. This penultimate set contained 24 apps. We decided not to analyze Google Messages because it is owned by Google and, therefore, there is no notion of third-party leakage in that app; Google runs the infrastructure that provides the push notifications. We also excluded Leo Messenger, which appeared to aggregate other messaging apps and did not have messaging functionality in its own right, as well as Gap Messenger, for which we were unable to register. Therefore, the final set contained 21 apps.

4.2 App Analysis

We performed dynamic and static analysis on each secure messaging app in our data set to learn how the usage of FCM impacted user privacy. Specifically, did the app naïvely leverage the default FCM behavior and include plaintext user content? Or, did the app use specific techniques to protect the privacy of user messages above and beyond what FCM offers by default? (For example, by integrating the Capillary library [13] mentioned in Google's blog.)

Data Types. In our analysis, we searched for specific data types that we expected to appear in the content of push notifications. To compile the list of these data types, we started with the data types defined and used by Google's privacy labels [35], which also enabled us to compare observed practices with the privacy labels declared by each app's developer. As we present in Section 5, we found evidence of the following data types being leaked to Google: (1) *Device or other IDs*, (2) *User IDs*, (3) *Name*, (4) *Phone Number*, and (5) *Message Contents*. Unlike (1) to (4), the contents of communications are afforded additional protections in many jurisdictions due to their sensitive nature.[11] We present additional information about these data types in Appendix A.

We performed our analysis in early 2023 with an instrumented version of Android 12, at a time when the majority of users (more than 85%) had Android version 12 or below installed on their phones [75]. Using a Pixel 3a phone, we installed each app from Google Play Store and saved its Android package (APK) files and privacy disclosures. We also created test accounts where necessary. We then used dynamic analysis to identify what personal information got leaked to FCM and static analysis to understand what strategies apps used to protect user privacy.

Data Leakages. We used dynamic analysis to record the contents of a push notification after our device received it from the FCM server. We instrumented the `keySet()` method of the standard `BaseBundle` class [32], which gets called by the FCM SDK, and logged the contents of the `Bundle` only if it contained the default keys in a push notification, such as "google.message_id." Additionally, we used Frida [29] to instrument the `handleIntent` method of `FirebaseMessagingService` [34], which listens and receives FCM push notifications as broadcasts from Google Play Services. This method then delivers push notification contents to app-specific callback methods (e.g., `onMessageReceived`), which allow the app to handle and display push messages as notifications to users.

The main goal was to trigger a push notification so that the resulting payload sent from Google's FCM server to our test device could be recorded (connection 3 in Figure 2). We installed each app on two devices and triggered push notifications by sending messages from one device to another. On the recipient's Pixel 3a device, we recorded the push notification contents as they were received by the app using the instrumented methods.

[7] 15 U.S.C. §45.
[8] https://github.com/facundoolano/google-play-scraper
[9] "messaging," "chat," "internet," "friend," and "in touch."
[10] "SMS," "browser," "VPN," "recover," and "voicemail."
[11] E.g., Title I of the Electronic Communications Privacy Act of 1986 (ECPA) [90].

Privacy Strategy. The push notifications that we observed fell into one of the following three categories:

(1) **No Protection.** The FCM push notification contained all of the information (i.e., username and message contents) that the app uses to display the notification.
(2) **Some Protection.** The FCM push notification contained some personal information but, notably, did not include the displayed message contents in plaintext.
(3) **Full Protection.** The FCM push notification did not contain any personal information, and any additional fields were empty or always contained unique values (i.e., not corresponding to any persistent identifiers).

For the first case, we simply assumed that the app does not use any privacy protection strategies. For the latter two cases, determining the strategy was often straightforward. For instance, Skype (in secret chat) included `EndToEndEncryption` as the value for the `messagetype` key, while Session included the `ENCRYPTED_DATA` key with a value corresponding to an encoded message. Signal, on the other hand, received FCM push notifications that only contain the empty field `notification` without any other content.

To validate the identified strategies, we performed static analysis. We first decompiled the APKs for each closed-source app using the `jadx`[12] Dex to Java decompiler. Analyzing obfuscated code was often complex. We searched for `FirebaseMessagingService` to find services that extend it. We then examined the code of these services to see how they implement the `onMessageReceived` method, which gets invoked by the FCM SDK whenever the app running on the client device receives a push notification. Crucially, the SDK also passes a hash table of type `RemoteObject` containing information necessary to display the notification to the user and, optionally, a data payload to perform any custom functions triggered by the receipt of a notification.

We tried to determine whether the push notifications contain sensitive content by observing the strings defined in code and used in the names of the keys or in print statements. We then traced the message and any variables assigned to the sensitive content until we reached the code for displaying the notification to the user. Appendix B includes the questions we used to analyze the source code of apps in our data set.

4.3 Privacy Disclosure Analysis

The final phase of our analysis involved comparing the claims that app developers made in their privacy disclosures to the ground truth that we observed from our dynamic and static analysis. Therefore, we focused on the 11 app developers that we observed including personal information in the push notifications sent via Google's FCM (§ 5). We wanted to determine whether a user could reasonably conclude that the app guarantees the security and privacy of their personal information based on the information presented by the app vendor in their Play Store description, official website, marketing and promotional materials, and other documentation. Moreover, we wanted to understand whether developers disclose the sharing of personal information for the purposes of providing push notifications in their privacy policies.

To achieve this, several researchers from our team first located statements by app vendors that talk about the security and privacy of messages. We also determined whether the apps (that we observed sharing personal information with Google) claimed to support end-to-end encryption by default, potentially misleading the users about the privacy of their messages or their metadata. Finally, we read each privacy policy to determine whether they stated that the particular types of personal information we observed might be shared with service providers for the purpose of app functionality. If it did, we further recorded whether the privacy policy listed the specific service providers or the specific types of data shared for the purpose of app functionality, which we compared against the results of our app analysis. By cross-referencing the different sources of information about an app's privacy practices, we aimed to build a holistic picture of how each developer frames the privacy risks associated with use of their app. We saved static copies of each privacy disclosure and the privacy policies using the Internet Archive's Wayback Machine [11].

4.4 Ethical Research

Our work involves reverse-engineering the client apps of popular Android secure instant messengers in order to glean the types of information being leaked to Google's FCM servers in push notifications. We performed our analysis by running each app on our test devices, with test accounts, on a segmented and private network, and observing both the network traffic that resulted and, when that network traffic did not reveal personal information, the static code. We were only interested in observing the leakage of personal information pertaining to our test devices; we did not interact with other app users nor did we make any attempts to obtain personal information of other users. Our study did not involve human subjects, nor did it involve unauthorized access to protected systems.

As we discuss in Section 5, we found inconsistencies between the observed app behavior and promises made by developers of several apps from our data set (see also Table 1). We disclosed our findings to those developers to ensure these inconsistencies can be addressed promptly (see § 7 for a further discussion).

5 RESULTS

We present findings from our analysis of secure messaging apps, including the personal information observed being shared with Google's FCM servers and the mitigation strategies employed by apps to prevent such leakage. Additionally, we analyzed statements made by app developers to determine whether they make any privacy or security guarantees and whether they disclose the sharing of personal information for push notifications.[13]

5.1 App Analysis

We found that almost all analyzed applications used FCM. Of the popular secure messaging apps that we identified, 20 of 21 apps relied on FCM to deliver push notifications to users. One exception among those apps was Briar messenger, which prompted the user to enable unrestricted battery usage, allowing the app to poll for new messages in the background. (Several other apps in our dataset

[12] https://github.com/skylot/jadx

[13] Supplemental materials are available at https://github.com/blues-lab/fcm-app-analysis-public.

App	Privacy Strategy	Message Content	Device IDs	User IDs	Name	Phone #
Skype (default)	None	●	●	●	●	○
(secret chat)	E2EE	○	●	●	●	○
Snapchat	E2EE	○	○	●	●	○
Viber	Push-to-Sync	○	●	●	○	●
LINE	E2EE	○	○	○	●	○
Discord	None	●	○	●	●	○
WeChat	None	●	○	●	●	○
JusTalk	None	●	○	●	●	○
SafeUM	E2EE	○	○	●	○	○
YallaChat	E2EE	○	○	●	●	○
Comera	Push-to-Sync	○	○	●	○	●
Wire	Push-to-Sync	○	●	●	○	○

Table 1: This table contains all analyzed apps, for which we observed personal information leakage to FCM servers in the process of delivering push notifications. The specific observed category of data is indicated by ● (evidence) and ○ (no evidence).

also prompted us to enable unrestricted battery usage, however, those apps still relied on FCM.) Since our study focuses on FCM, we excluded Briar and analyzed only those applications that relied on FCM to deliver push notifications.

Of the 20 apps we analyzed, 11 included personal information in data sent to Google via FCM such that that data was visible to Google. All 11 apps leaked message metadata, including device and app identifiers (3 apps), user identifiers (10 apps), the sender's or recipient's name (7 apps), and phone numbers (2 apps). More alarmingly, we observed 4 apps—which have cumulative installs in excess of one billion—leak message contents. We present information about the observed practices in Table 1.

It is worth noting that not all of the observed behaviors here are necessarily *undisclosed sharing*. Undisclosed sharing occurs when data we observed being shared from our static and/or dynamic analysis was not disclosed in the privacy disclosures we analyzed. Whether the observed behaviors do constitute undisclosed sharing depends on the findings from our privacy disclosure analysis, discussed below (§5.3).

5.2 Mitigation Strategies

Of the 16 apps that did not send message contents to Google.[14] our static analysis revealed two general mitigation strategies described below: end-to-end encryption and push-to-sync. Ultimately, we observed 9 apps out of 16 employ either end-to-end encryption or push-to-sync strategies to prevent leaking any personal information to Google via FCM. The remaining 7 apps still leaked metadata, but not the message contents. See Table 2 for more information.

End-to-End Encryption. We determined that 8 apps employed an end-to-end encryption strategy to prevent privacy leakage to Google via FCM. In this strategy, when the user launches the app for the first time, the app provisions a keypair and does a secure key exchange between the user's device and the app's server. The app will then develop a session key that it can use to decrypt messages from the server. The server encrypts messages it sends using the session key before it goes to FCM.

[14]Skype used e2e encryption to protect message contents only in secret chats, which is not the default.

```
firebase:message:10276:START:{
    google.delivered_priority=high,
    google.sent_time=1677001395829,
    google.ttl=2419200,
    google.original_priority=high,
    from=312334754206,
    google.message_id=0:1677001395846147…,
    notification=,
    google.c.sender.id=312334754206
}
```

Figure 4: Example payload from within the `RemoteMessage` object received by the Signal app. Note the empty notification field, indicating the correct usage of the *push-to-sync* notification strategy.

As depicted in Table 2, of the 8 apps that utilized the end-to-end encryption (e2e) strategy, only 4 (Facebook Messenger, Telegram, Session, and KakaoTalk) did not leak *any* personal information to Google via FCM. The remaining 4 (Snapchat, SafeUM, YallaChat, and LINE) still leaked metadata, including user identifiers (3 apps) and names (3 apps).

Push-to-Sync. We observed 8 apps employ a push-to-sync strategy to prevent privacy leakage to Google via FCM. In this mitigation strategy, apps send an empty (or almost empty) push notification to FCM. Some apps, such as Signal, send a push notification with no data (aside from the fields that Google sets; see Figure 4). Other apps may send an identifier (including, in some cases, a phone number). This push notification tells the app to query the app server for data, the data is retrieved securely by the app, and then a push notification is populated on the client side with the unencrypted data. In these cases, the only metadata that FCM receives is that the user received some message or messages, and when that push notification was issued. Achieving this requires sending an additional network request to the app server to fetch the data and keeping track of identifiers used to correlate the push notification received on the user device with the message on the app server.

App	Version	Uses FCM?	Privacy Strategy	Observed Data Leakage	Min Installs (millions)
Facebook Messenger	v403.1.0.17.106	●	e2ee	□	5,000
WhatsApp	v2.23.12.78	●	Push-to-Sync	□	5,000
Skype	v8.93.0.408	●	**none** (default) e2ee (secret chat)	⊠	1,000
Snapchat	v12.28.0.22	●	e2ee	⊠	1,000
Telegram	v9.4.4	●	e2ee	□	1,000
Viber	v19.4.0.0	●	Push-to-Sync	⊠	1,000
LINE	v13.4.2	●	e2ee	⊠	500
Discord	v172.24	●	**none**	⊠	100
Kakao Talk	v10.0.7	●	e2ee	□	100
Kik	v15.50.1.27996	●	Push-to-Sync	□	100
Signal	v6.11.7	●	Push-to-Sync	□	100
WeChat	v8.0.30	●	**none**	⊠	100
JusTalk	v8.6.10	●	**none**	⊠	10
SafeUM	v1.1.0.1548	●	e2ee	⊠	5
YallaChat	v1.4.2	●	e2ee	⊠	5
Briar	v1.4.23	○	Polling	□	1
Comera	v4.0.1	●	Push-to-Sync	⊠	1
Element	v1.5.22	●	Push-to-Sync	□	1
Session	v1.16.7	●	e2ee	□	1
Threema	v5.0.6	●	Push-to-Sync	□	1
Wire	v3.82.38	●	Push-to-Sync	⊠	1
				TOTAL installs	**15,026**

Table 2: Our data set of analyzed apps. Usage of Firebase Cloud Messaging (FCM) is indicated by ●(does use) and ○(does not use). Whether or not an app leaked personal information to FCM is indicated by □(no evidence) and ⊠(evidence). See Table 1 for details on which personal data is leaked by apps marked with ⊠. Apps are sorted by minimum install count and alphabetically by app name.

As detailed in Table 2, only 5 (Whatsapp, Signal, Threema, Element, and Kik) did not leak any personal information to Google. The remaining 3 (Viber, Wire, and Comera) leaked metadata, including user identifiers (all 3 apps), device and app identifiers (2 apps), and phone numbers (2 apps).

5.3 Privacy Disclosure Analysis

We analyzed privacy disclosures for the 11 apps that included personal information in the push notifications sent via Google's FCM. One of our aims was to determine whether a user could reasonably conclude that the app guarantees the security and privacy of their personal information based on the information presented by the app vendor in their Play Store description, official website, marketing and promotional materials, and other documentation. Table 3 provides details for each app.

Marketing Claims. First, we discuss the 4 apps that leaked the actual contents of push notification messages: Skype, WeChat, Discord, and JusTalk. We found that out of these four apps, only JusTalk claimed to be end-to-end secure, stating: "All users' personal information (including calling and messaging data) is end-to-end encrypted and is split into multiple random paths which ensure it can't be monitored or saved by servers. Moreover, all the personal data is never shared with any third party. Enjoy safe and free calls" [42]. Nevertheless, we clearly observed the contents of our messages being sent without end-to-end encryption via FCM's servers while delivering push notifications (see Figure 5).

```
firebase:message:10279:START:{
    google.delivered_priority=high,
    google.sent_time=1677010922128,
    google.ttl=2419200,
    google.original_priority=high,
    resend=0,
    MtcImTextKey=Hello Dustin! How are you doing?,
    MtcImTimeKey=1677010922031,
    MtcImUserDataKey={},
    MtcImInfoTypeKey=Text,
    from=144552557193,
    toUid=9999_43035938,
    google.message_id=0:1677010922135234%...,
    MtcImLabelKey=P2P/9999_43036012,
    MtcImDisplayNameKey=Charlotte,
    google.c.sender.id=144552557193,
    MtcImMsgIdKey=0,
    MtcImImdnIdKey=97866160-0e6a-495a-9932...,
    MtcImSenderUidKey=9999_43036012
}
```

Figure 5: Payload contained inside the `RemoteMessage` object received by JusTalk. Note the `MtcImTextKey` and `MtcImDisplayNameKey`, which contain the unencrypted message contents and username, respectively.

Although the three remaining apps do not claim to employ end-to-end encryption, both WeChat and Discord made statements about their concern for privacy. For instance, WeChat said in their Play Store description: "- BETTER PRIVACY: Giving you the highest level of control over your privacy, WeChat is certified by TRUSTe" [62]. Although Skype does not reference secure messaging for their normal (default) chat functionality, they promise that "Skype private conversations uses the industry standard Signal Protocol, allowing you to have end-to-end encrypted Skype audio calls, send text messages, image, audio, and video files" [56]. Although we did not observe the content of the message being leaked when testing Skype's private conversation feature, we still observed the app leaking device IDs, user IDs, and names via Google's FCM.

For the remaining 7 apps that did not leak message contents, we observed each of these apps make claims that could lead users to believe that the apps do not share any personal information with anyone and, except for Snapchat, claimed to be end-to-end encrypted. For instance, SafeUM messenger put it plainly: "[w]e never share your data with anyone. Never" [67].

Privacy Policies. We additionally read each privacy policy to understand whether developers disclosed the sharing of personal information for the purposes of providing push notifications. We found that all 11 apps that shared personal information with Google's FCM servers stated that personal user data may be shared with service providers (such as FCM) for the purpose of app functionality. However, only two apps (JusTalk and YallaChat) enumerated the types of personal information shared with such service providers, which did not cover the types of information we observed being shared, namely user IDs and names (for both apps) and message contents (for JusTalk, as discussed above). Furthermore, three apps (Viber, WeChat and Comera) did not specify which companies serve as their service providers. Out of the remaining 8 apps, only 4 mentioned Google in the context of push notifications and/or FCM.

Given that only YallaChat included information about the types of data shared with Google's FCM, we were unable to determine whether the specific data types we observed being shared would be covered by these statements or not. For instance, Viber's privacy policy stated, without giving any specifics: "[w]e may disclose your Personal Information to a contractor or service provider for a business purpose. The types of personal information we share for a business purpose, vary, depending on the purpose and the function provided by the third party to whom we disclose such information" [91]. While these statements may technically address personal data sharing in the context of push notifications, they do not meaningfully inform users about *what* information pertaining to them is being shared and *with whom*.

6 DISCUSSION

The democratization of mass communications via the Internet has created a new paradigm in which anyone can have a platform to send a message. Consequently, anyone can now become a software engineer and distribute software worldwide. By and large, this is a good thing. However, it raises issues of professional responsibility that have long been addressed by other more mature branches of engineering. In most jurisdictions, one cannot simply decide to become a civil engineer and erect a multi-story building. Due to the inherent safety risks—to the individual, neighbors, and society—most jurisdictions require that plans be submitted for approval. In granting that approval, the plans are first checked for conformance with building codes, which have been set (and periodically revised) by professional societies with deep expertise. Once plans are approved, multiple levels of oversight still occur: at various steps during construction, building inspectors confirm that both the plans have been followed and that no other safety issues have been identified. Moreover, after construction has been completed, governments are empowered to continually monitor for code violations, going so far as to condemn structures that pose safety hazards. Of course, there is a reason for this oversight: building codes are written in blood.

In the past decade or two, software engineering as a discipline has only just begun to reckon with the complex sociotechnical issues relating to harm and liability. While the collapse of a building is likely to be more lethal than the inappropriate exfiltration of sensitive user information, the latter may still pose risks to user safety—even lethal ones. We chose to examine secure messaging apps in this study because they can often embody these risks: online messaging apps are increasingly being used by activists living in oppressive regimes [85], who may find themselves in serious jeopardy if their communications are inappropriately revealed. In this specific instance, the inappropriate disclosure of users' communication and metadata does not require malice on the part of a service provider for harm to come to the user. By nature of such data collection, the service provider exposes the user to legal processes: this may result in data the user legitimately did not believe to exist coming into the hands of governments and private actors. We emphasize that this risk is not merely theoretical; as previously noted, U.S. Senator Ron Wyden published a letter that confirms that government agencies do, in fact, collect user information by demanding push notification records from Google and Apple [100].

Our analysis found that several prevalent secure messaging apps—which imply that they will not share certain information with third parties—do indeed share that information in plaintext with Google via FCM (see Table 1). We found evidence of undisclosed data leakage to FCM in apps that account for over 2 billion installs. Users of these apps are likely unaware of these data leakages: some of the privacy disclosures made by these apps often explicitly promise *not* to share such personal information with third parties, whereas others were so vaguely written that it was unclear whether these behaviors are being disclosed (and how they might comport in consumers' minds with the companies' marketing materials that imply messaging data will be kept private). Consequently, consumers may have a false sense of security when using these apps for communicating. The undisclosed leakage of communication contents can harm users and potentially even innocent bystanders who may be mentioned in communications.

6.1 Recommendations

Just as a contractor or owner-builder is ultimately responsible for the adherence to local building codes and the risks associated with deviations from them, software developers publishing apps for public usage are responsible for the behaviors of those apps. This

App		E2EE	S/P	Discloses PI Sharing	Discloses Companies	Discloses Shared PI
Skype	(default)	○	○	●	●	○
	(secret chat)	●	●	●	●	○
Snapchat		○	●	●	●	○
Viber		●	●	●	○	○
LINE		●	●	●	●	○
Discord		○	●	●	●	○
WeChat		○	●	●	○	○
JusTalk		●	●	●	●	●
SafeUM		●	●	●	●	○
YallaChat		●	●	●	●	●
Comera		●	●	●	○	○
Wire		●	●	●	●	○

Table 3: This table contains information about the disclosures made by developers of apps, for which we observed information leakage to FCM. ● indicates that we found evidence (or ○ if not) for each of the following statements: (E2EE) developer states the app uses end-to-end encryption, (S/P) developer makes security or privacy-specific claims in the Google Play Store description or on their official websites, (discloses PI sharing) developer discloses in their privacy policy the sharing of personal information to service providers for app functionality purposes, (discloses companies) if the disclosure includes names of companies and (discloses shared PI) if the disclosure includes specific types of personal information.

responsibility includes verifying that third-party components function as expected and that the ultimate behavior of the app is in accordance with platform guidelines, the developer's disclosures, and applicable laws/regulations. The use of these third-party components is not unique to software engineering: other branches of engineering generally involve complex supply chains, yet there is often a great deal of oversight. When Airbus builds a plane, they may use engines from Rolls-Royce or electronics from Siemens; but in addition to simply specifying the specifications and tolerances that Airbus expects these components to conform to, they nonetheless validate those third-party components by launching chickens at them at 600+ km/h (amongst other validation tests) [98]. Such integration validations rarely exist for software *in practice*, despite being recommended for nearly half a century now [28]. Indeed, while we have no reason to believe that misleading or confusing security and privacy claims are the result of malice, we believe that the poor privacy practices that we document in this paper could have been discovered and mitigated by the developers had they inspected the traffic sent and received by their applications during quality assurance processes. Thus, we offer recommendations to different stakeholders on ways to address the identified security and privacy issues.

6.1.1 App Developers. As the parties ultimately responsible for their apps, app developers should perform the type of dynamic analysis that we performed in this study as part of each and every release cycle. This will help to ensure that users' personal data flows in accordance with reasonable expectations, applicable laws and regulations, as well as platform policies. However, the best way to ensure that push notifications do not leak sensitive user information is to avoid sending sensitive user information via FCM in the first place. We argue that developers should implement the push-to-sync approach: the developer's server should only send the app a unique notification ID via FCM, which can then be used to fetch the notification content from the developer's servers securely. Several developers correctly used the push-to-sync approach, which resulted in no personal data being leaked by those apps. Others should adopt this architecture in their apps.

6.1.2 Platforms and SDK Providers. At the same time, platform owners and SDK providers are well-positioned to identify and correct issues in their tools and highlight security and privacy risks in their documentation. For its part, Google provides an API that results in developers systematically making very similar privacy mistakes. This is not helped by Google's guidance, which instructs developers to "send as much data as possible in the FCM payload," and that if they want to do so securely, they must use an additional library [69]. This guidance departs from Google's own data minimization and secure-by-default principles [33] and recommendations from other push notification providers, such as Apple [10].

We argue that the availability of usable, secure push notifications libraries, including Google's Capillary [13], does not solve the underlying problem. Developers generally trust Google's security practices and are largely unaware of the risk of personal information leakage via push notifications. Furthermore, under current regulatory regimes, Google is not obligated to do anything about this: they provide a free API for developers, and despite the fact that using it to send messages *securely* admittedly takes additional non-obvious steps, there are no legal requirements that Google—or any other SDK provider—provide a secure-by-default API. Furthermore, as mentioned previously, Android app developers are effectively required to use Google's FCM to send push notifications for battery consumption reasons. We argue, therefore, that real-world change will require either applying regulatory pressure or other market-corrective forces on platform owners to enforce privacy-by-design principles for critical SDKs in the software supply chain, such as Google's FCM. Such a change would improve the privacy and security of nearly all Android apps, because the use of FCM to deliver push notifications on Android is nearly universal.

The use of these types of APIs also represents the classic usable security problem (wherein application developers are the "user"): the user is not qualified to be making the decisions that are forced upon them, whereas those forcing them to make these decisions are in a much better position to make those decisions on the users' behalf. Prior research shows that developers, despite being the party ultimately responsible for the behaviors of their software, are woefully unprepared to make these types of decisions [1, 4]. And thus, we are faced with a situation in which the parties most equipped to fix these types of problems (e.g., by creating more usable documentation that highlights security and privacy risks, making SDK settings secure by default, proactively auditing how their services are used in practice, etc.) are not incentivized to do so, whereas the parties who are ultimately responsible are generally incapable and do not understand their risks or responsibilities. As a result, this is fundamentally an economics problem concerning misaligned incentives [5]: in a perfect world, the responsibility for handling users' data responsibly would be placed upon those according to their abilities, shifted from those according to their needs [54]. This is not the world in which we currently live.

Yet, things are improving. In recent years, the U.S. Government has promoted the strategy of shifting the burden of software security away from individuals, small businesses, and local governments and onto the organizations that are most capable and best-positioned to reduce risks [82]. In line with this initiative, the U.S. Cybersecurity and Infrastructure Security Agency (CISA) and 17 U.S. and international partners published an update in August 2023 to joint guidance for implementing secure-by-design principles [21]. One secure product development practice, in particular, highlights the need to provide secure defaults for developers by "providing safe building blocks...known as 'paved roads' or 'well-lit paths.'" We believe that push notification providers can similarly apply privacy-by-design principles [60] to safeguard the privacy of users who cannot easily manage the risks.

Without correctly aligned incentives to motivate platforms and SDK providers to make their systems secure by default (including documentation that highlights security and privacy risks), developers will continue to be placed in this position and will continue to consistently make these types of mistakes. Thus, until software engineering becomes a more mature field with formalized oversight, validation, disclosure, and auditing procedures, these types of errors will proliferate, leaving end users at risk.

7 RESPONSIBLE DISCLOSURE

Responsible disclosure is a critical component of security and privacy research. We reported our substantive findings to the 11 app developers who leaked at least one personal data type to Google's FCM service. We tried contacting the developers via various contact methods, including formal bug bounty programs, emailing security teams, or failing that, general support contacts. The app developers for whom we could find contact information were sent a summary report on or before June 7, 2024. We received an acknowledgment of our email from 5 developers of the 11 we contacted.

At the time of publication, the remaining 6 app developers to whom we disclosed our findings had not replied; discussions are ongoing with several companies regarding how they should fix the identified issues. We look forward to continue engaging in productive conversations to help developers understand how to adapt their push message architectures to better protect user privacy.

8 LIMITATIONS

Many apps beyond secure messaging apps might send private data through push notifications. Our study only focused on secure messaging apps because most of them claim to focus on user privacy, thus, they would be among the most likely apps to take proactive steps to prevent the leakage of user data to FCM (and presumably users of these apps are more likely to believe that their communications are secure). We suspect that privacy leakage via Google FCM may be even more prevalent within apps in other contexts. Future work should look at both less popular secure messaging apps and apps in other contexts to observe to what extent, if at any, they mitigate the leakage of sensitive personal data to Google via FCM.

We also performed our analysis using an older Pixel 3a device running Android 12. We are unaware of any substantial changes in Android 13 and 14 that would have a material impact on our observed findings. Our device supported security updates and the installation of all the apps that we analyzed for this research. We ran these apps and received push notifications from FCM without observing any undesirable impact on app performance. Furthermore, at the time we began our analysis in early 2023, the majority of users (more than 85%) used Android version 12 or below [75]. While most people who use a mobile phone use an Android device, iOS also has a significant share of the mobile phone market and tends to bill itself as having more privacy-preserving practices. Future work can also explore whether private user data is leaked to Apple or other third parties via the push notification infrastructure available to developers in the iOS ecosystem.

We looked specifically at privacy leakage through push notifications that rely on FCM. As far as we know, FCM is also used in other applications, on Android and beyond; how this fact affects privacy leakage across other applications is not well understood. Future work could investigate the privacy implications of FCM across those applications. Within the Android ecosystem, there may exist other patterns or tools provided by Google or by other popular third-party libraries that also incur unexpected privacy leakage. Future work could look for such patterns beyond the Android platform, such as iOS, and identify how other ecosystem players like Apple and Google can craft a more trustworthy ecosystem to provide more privacy-preserving defaults to the broadest base of users.

> *"The personal and social consequences of any medium—that is, of any extension of ourselves—result from the new scale that is introduced into our affairs by each extension of ourselves, or by any new technology"*
>
> —Marshall McLuhan [55].

ACKNOWLEDGMENTS

This work was supported by the U.S. National Science Foundation under grant CCF-2217771, the Center for Long-Term Cybersecurity (CLTC) at U.C. Berkeley, the KACST-UCB Center of Excellence for Secure Computing, an NSERC Discovery Grant, and a grant from the Silicon Valley Community Foundation. We would especially like to thank the Office of U.S. Senator Ron Wyden for outreach that

inspired this work, as well as Chris Hoofnagle for early support and feedback, and of course, Refjohürs Lykkewe.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 154–171.

[2] Mansour Ahmadi, Battista Biggio, Steven Arzt, Davide Ariu, and Giorgio Giacinto. 2016. Detecting misuse of google cloud messaging in android badware. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*. 103–112.

[3] AirShip. 2023. Android SDK Setup. https://docs.airship.com/platform/mobile/setup/sdk/android/. (Accessed on 10/10/2023).

[4] Noura Alomar and Serge Egelman. 2022. Developers say the darnedest things: Privacy compliance processes followed by developers of child-directed apps. *Proceedings on Privacy Enhancing Technologies* 4, 2022 (2022), 24.

[5] R. Anderson. 2001. Why information security is hard - an economic perspective. In *Seventeenth Annual Computer Security Applications Conference*. 358–365. https://doi.org/10.1109/ACSAC.2001.991552

[6] 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 *28th USENIX security symposium (USENIX security 19)*. USENIX, Berkeley, CA, USA, 585–602.

[7] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. 2020. Actions Speak Louder than Words:Entity-Sensitive Privacy Policy and Data Flow Analysis with PoliCheck. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX, Berkeley, CA, USA, 985–1002.

[8] Apple. 2023. Notifications Overview. Apple Developer. https://developer.apple.com/notifications/.

[9] Apple. 2023. Push Token Requests. https://www.apple.com/legal/transparency/push-token.html. (Accessed on 06/01/2024).

[10] Apple Inc. 2023. Generating a remote notification . https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/generating_a_remote_notification. (Accessed on 10/10/2023).

[11] Internet Archive. 2023. Wayback Machine. https://archive.org/. (Accessed on 10/10/2023).

[12] Kayce Basques and Matt Gaunt. 2023. Push notifications overview. https://web.dev/articles/push-notifications-overview. (Accessed on 10/10/2023).

[13] Android Developers Blog. 2018. Project Capillary: End-to-end encryption for push messaging, simplified. https://android-developers.googleblog.com/2018/06/project-capillary-end-to-end-encryption.html. (Accessed on 10/10/2023).

[14] Duc Bui, Kang G Shin, Jong-Min Choi, and Junbum Shin. 2021. Automated Extraction and Presentation of Data Practices in Privacy Policies. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2021, 2 (2021), 88–110.

[15] L. Cavallaro, P. Saxena, and R. Sekar. 2008. On the Limits of Information Flow Techniques for Malware Analysis and Containment. In *Proc. of DIMVA*. Springer-Verlag, 143–163. http://dx.doi.org/10.1007/978-3-540-70542-0_8

[16] Ann Cavoukian. 2009. Privacy by design. (2009).

[17] Yangyi Chen, Tongxin Li, XiaoFeng Wang, Kai Chen, and Xinhui Han. 2015. Perplexed messengers from the cloud: Automated security analysis of push-messaging integrations. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1260–1272.

[18] U.S. Federal Trade Commission. 2021. Flo Health, Inc. https://www.ftc.gov/legal-library/browse/cases-proceedings/192-3133-flo-health-inc.

[19] U.S. Federal Trade Commission. 2024. Avast, Ltd. https://www.ftc.gov/system/files/ftc_gov/pdf/Complaint-Avast.pdf.

[20] Cox, Joseph. 2023. Here's a Warrant Showing the U.S. Government is Monitoring Push Notifications. https://www.404media.co/us-government-warrant-monitoring-push-notifications-apple-google-yahoo/. (Accessed on 06/01/2024).

[21] Cybersecurity and Infrastructure Security Agency (CISA). 2023. Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software. https://www.cisa.gov/sites/default/files/2023-10/SecureByDesign_1025_508c.pdf. (Accessed on 06/01/2024).

[22] Samsung Electronics. 2023. Samsung Push Service. https://play.google.com/store/apps/details?id=com.sec.spp.push. (Accessed on 06/01/2024).

[23] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. 2010. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proc. of the 9th USENIX conference on Operating systems design and implementation (OSDI)*. 393–407.

[24] Ming Fan, Le Yu, Sen Chen, Hao Zhou, Xiapu Luo, Shuyue Li, Yang Liu, Jun Liu, and Ting Liu. 2020. An empirical evaluation of GDPR compliance violations in Android mHealth apps. In *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*. IEEE, New York, NY, USA, 253–264.

[25] Federal Trade Commision (FTC). 2020. FTC Requires Zoom to Enhance its Security Practices as Part of Settlement. https://www.ftc.gov/news-events/news/press-releases/2020/11/ftc-requires-zoom-enhance-its-security-practices-part-settlement. (Accessed on 01/01/2024).

[26] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. 2012. Android permissions: user attention, comprehension, and behavior. In *Proceedings of the 8th Symposium on Usable Privacy and Security* (Washington, D.C.) *(SOUPS '12)*. ACM, New York, NY, USA, Article 3, 14 pages. https://doi.org/10.1145/2335356.2335360

[27] European Union Agency for Cybersecurity (ENISA). 2023. Engineering Personal Data Sharing. https://www.enisa.europa.eu/publications/engineering-personal-data-sharing. (Accessed on 06/01/2024).

[28] Frederick P. Brooks, Jr. 1975. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley.

[29] Frida. 2022. https://frida.re/.

[30] C. Gibler, J. Crussell, J. Erickson, and H. Chen. 2012. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *Proc. of the 5th international conference on Trust and Trustworthy Computing (TRUST)*. Springer-Verlag, 291–307.

[31] GizChina. 2023. HARMONYOS IS NOW FIRMLY THE THIRD LARGEST MOBILE PHONE OPERATING SYSTEM. https://www.gizchina.com/2023/05/20/harmonyos-is-now-firmly-the-third-largest-mobile-phone-operating-system/. (Accessed on 01/01/2024).

[32] Google. 2023. BaseBundle. Android Developers. https://developer.android.com/reference/android/os/BaseBundle.

[33] Google. 2023. Design for Safety. Google Developers. https://developer.android.com/quality/privacy-and-security.

[34] Google. 2023. FirebaseMessagingService. https://firebase.google.com/docs/reference/android/com/google/firebase/messaging/FirebaseMessagingService. (Accessed on 06/01/2024).

[35] Google. 2023. Play Console Help: Provide information for Google Play's Data safety section. https://support.google.com/googleplay/android-developer/answer/10787469. (Accessed on 06/01/2024).

[36] Google for Developers. 2024. About FCM messages. Developer documentation for Firebase. https://firebase.google.com/docs/cloud-messaging/concept-options.

[37] M. I. Gordon, D. Kim, J. Perkins, Gilhamy, N. Nguyenz, and M. Rinard. 2015. Information-Flow Analysis of Android Applications in DroidSafe. In *Proc. of NDSS Symposium*.

[38] Marit Hansen, Meiko Jensen, and Martin Rost. 2015. Protection goals for privacy engineering. In *2015 IEEE Security and Privacy Workshops*. IEEE, 159–166.

[39] Hamza Harkous, Kassem Fawaz, Rémi Lebret, Florian Schaub, Kang G Shin, and Karl Aberer. 2018. Polisis: Automated analysis and presentation of privacy policies using deep learning. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX, Berkeley, CA, USA, 531–548.

[40] Harwell, Drew and Schaffer, Aaron. 2024. The FBI's new tactic: Catching suspects with push alerts. https://www.washingtonpost.com/technology/2024/02/29/push-notification-surveillance-fbi/. (Accessed on 06/01/2024).

[41] Sangwon Hyun, Junsung Cho, Geumhwan Cho, and Hyoungshick Kim. 2018. Design and analysis of push notification-based malware on android. *Security and Communication Networks* 2018 (2018).

[42] JusTalk. 2023. Is it safe to use JusTalk? https://web.archive.org/web/20230407183707/https://justalk.com/support/general/g6. (Accessed on 10/10/2023).

[43] P. G. Kelley, L. F. Cranor, and N. Sadeh. 2013. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 3393–3402.

[44] J. Kim, Y. Yoon, K. Yi, and J. Shin. 2012. ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications. *IEEE Workshop on Mobile Security Technologies (MoST)* (2012).

[45] Simon Koch, Malte Wessels, Benjamin Altpeter, Madita Olvermann, and Martin Johns. 2022. Keeping privacy labels honest. *Proceedings on Privacy Enhancing Technologies* 4, 486-506 (2022), 2–2.

[46] Konev, Max. 2022. Statement on the Reuters Story Regarding Pushwoosh. https://blog.pushwoosh.com/blog/statement-on-the-reuters-story-regarding-pushwoosh/. (Accessed on 06/01/2024).

[47] Hayoung Lee, Taeho Kang, Sangho Lee, Jong Kim, and Yoonho Kim. 2014. Punobot: Mobile botnet using push notification service in android. In *Information Security Applications: 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers 14*. Springer, 124–137.

[48] Tongxin Li, Xiaoyong Zhou, Luyi Xing, Yeonjoon Lee, Muhammad Naveed, XiaoFeng Wang, and Xinhui Han. 2014. Mayhem in the push clouds: Understanding and mitigating security hazards in mobile push-messaging services. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 978–989.

[49] Thomas Linden, Rishabh Khandelwal, Hamza Harkous, and Kassem Fawaz. 2018. The privacy policy landscape after the GDPR. *arXiv preprint arXiv:1809.08396* (2018), 1–18.

[50] Tianming Liu, Haoyu Wang, Li Li, Guangdong Bai, Yao Guo, and Guoai Xu. 2019. Dapanda: Detecting aggressive push notifications in android apps. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 66–78.

[51] Pierpaolo Loreti, Lorenzo Bracciale, and Alberto Caponi. 2018. Push attack: binding virtual and real identities using mobile push notifications. *Future Internet* 10, 2 (2018), 13.

[52] Jiadong Lou, Xiaohan Zhang, Yihe Zhang, Xinghua Li, Xu Yuan, and Ning Zhang. 2023. Devils in Your Apps: Vulnerabilities and User Privacy Exposure in Mobile Notification Systems. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 28–41.

[53] Mary Madden. 2014. Public Perceptions of Privacy and Security in the Post-Snowden Era. Pew Research Center. https://www.pewresearch.org/internet/2014/11/12/public-privacy-perceptions/.

[54] Karl Marx. 1875. *Critique of the Gotha program*.

[55] Marshall McLuhan. 1964. Understanding Media. (1964).

[56] Microsoft. 2023. What are Skype Private Conversations? https://web.archive.org/web/20230606085952/https://support.skype.com/en/faq/fa34824/what-are-skype-private-conversations. (Accessed on 10/10/2023).

[57] Ehimare Okoyomon, Nikita Samarin, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, Irwin Reyes, Álvaro Feal, Serge Egelman, et al. 2019. On the ridiculousness of notice and consent: Contradictions in app privacy policies. In *Workshop on Technology and Consumer Protection (ConPro 2019), in conjunction with the 39th IEEE Symposium on Security and Privacy*. IEEE, New York, NY, USA.

[58] OneSignal. 2023. Firebase Cloud Messaging (FCM) Compared to OneSignal. https://web.archive.org/web/20230603040346/https://onesignal.com/blog/firebase-vs-onesignal/. (Accessed on 10/10/2023).

[59] OneSignal. 2023. What is a push notifications service and how does it work? https://onesignal.com/blog/what-is-a-push-notifications-service-and-how-does-it-work/. (Accessed on 2/23/24).

[60] Frank Pallas, Katharina Koerner, Isabel Barberá, Jaap-Henk Hoepman, Meiko Jensen, Nandita Rao Narla, Nikita Samarin, Max-R Ulbricht, Isabel Wagner, Kim Wuyts, et al. 2024. Privacy Engineering From Principles to Practice: A Roadmap. *IEEE Security & Privacy* 22, 2 (2024), 86–92.

[61] James Pearson and Marisa Taylor. 2022. Russian software disguised as American finds its way into U.S. Army, CDC apps. https://www.reuters.com/technology/exclusive-russian-software-disguised-american-finds-its-way-into-us-army-cdc-2022-11-14/. (Accessed on 06/01/2024).

[62] Google Play. 2023. WeChat: About this app. https://web.archive.org/web/20230323082225/https://play.google.com/store/apps/details?id=com.tencent.mm&hl=en_US&gl=US. (Accessed on 10/10/2023).

[63] Pusher. 2023. Configure FCM. https://pusher.com/docs/beams/getting-started/android/configure-fcm/. (Accessed on 10/10/2023).

[64] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill. 2017. Studying TLS usage in Android apps. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 350–362.

[65] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, Serge Egelman, et al. 2018. "Won't somebody think of the children?" examining COPPA compliance at scale. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018, 3 (2018), 63–83.

[66] 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 *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 150–157.

[67] SafeUM. 2023. Privacy Policy. https://web.archive.org/web/20230220213832/https://safeum.com/privacypolicy.html. (Accessed on 10/10/2023).

[68] Nikita Samarin, Shayna Kothari, Zaina Siyed, Oscar Bjorkman, Reena Yuan, Primal Wijesekera, Noura Alomar, Jordan Fischer, Chris Hoofnagle, and Serge Egelman. 2023. Lessons in VCR Repair: Compliance of Android App Developers with the California Consumer Privacy Act (CCPA). *arXiv preprint arXiv:2304.00944* (2023).

[69] Jingyu Shi. 2023. Notifying your users with FCM. https://android-developers.googleblog.com/2018/09/notifying-your-users-with-fcm.html. (Accessed on 10/10/2023).

[70] Signal. 2023. Grand jury subpoena for Signal user data, Central District of California (again!). https://web.archive.org/web/20230921202338/https://signal.org/bigbrother/cd-california-grand-jury/. (Accessed on 10/10/2023).

[71] Signal. 2023. Signal. https://signal.org/. (Accessed on 10/10/2023).

[72] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. 2016. Toward a framework for detecting privacy policy violations in android application code. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, New York, NY, USA, 25–36.

[73] Ivan Slobozhan, Tymofii Brik, and Rajesh Sharma. 2023. Differentiable characteristics of Telegram mediums during protests in Belarus 2020. *Social Network Analysis and Mining* 13, 1 (2023), 19.

[74] Adam Smith. 1776. *An Inquiry Into the Nature and Causes of the Wealth of Nations*. Strahan and Cadell, London, UK. https://books.google.com/books?id=mt1SAAAAcAAJ

[75] StatCounter Global Stats. 2023. Android Version Market Share Worldwide. https://gs.statcounter.com/android-version-market-share/all/worldwide/2023. (Accessed on 06/01/2024).

[76] Anne Stopper and Jen Caltrider. 2023. See no evil: Loopholes in Google's data safety labels keep companies in the clear and consumers in the dark. mozilla foundation.

[77] J. Tan, K. Nguyen, M. Theodorides, H. Negron-Arroyo, C. Thompson, S. Egelman, and D. Wagner. 2014. The Effect of Developer-Specified Explanations for Permission Requests on Smartphone User Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

[78] Telegram. 2023. Telegram Messenger. https://telegram.org/. (Accessed on 10/10/2023).

[79] Telegram-FOSS on GitHub. 2024. Notifications. https://github.com/Telegram-FOSS-Team/Telegram-FOSS/blob/master/Notifications.md. (Accessed on 06/01/2024).

[80] The Drum. 2023. WhatsApp's 3D billboard touts privacy features. https://www.thedrum.com/news/2022/10/10/whatsapp-s-3d-billboard-touts-privacy-features. (Accessed on 10/10/2023).

[81] The Verge. 2023. Now Mark Zuckerberg's making fun of Apple for iMessage, too. https://www.theverge.com/2022/10/17/23409018/mark-zuckerberg-meta-whatsapp-imessage-privacy-security-ads. (Accessed on 10/10/2023).

[82] The White House. 2023. National Cybersecurity Strategy. https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf. (Accessed on 06/01/2024).

[83] C. Thompson, M. Johnson, S. Egelman, D. Wagner, and J. King. 2013. When It's Better to Ask Forgiveness than Get Permission: Designing Usable Audit Mechanisms for Mobile Permissions. In *Proceedings of the 2013 Symposium on Usable Privacy and Security (SOUPS)*.

[84] L. Tsai, P. Wijesekera, J. Reardon, I. Reyes, S. Egelman, D. Wagner, N. Good, and J. Chen. 2017. Turtle Guard: Helping Android Users Apply Contextual Privacy Preferences. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. USENIX Association, Santa Clara, CA, 145–162. https://www.usenix.org/conference/soups2017/technical-sessions/presentation/tsai

[85] Zeynep Tufekci. 2017. *Twitter and tear gas: The power and fragility of networked protest*. Yale University Press.

[86] UnifiedPush. 2023. UnifiedPush. https://unifiedpush.org/. (Accessed on 10/10/2023).

[87] United States District Court for the Central District of California. 2022. Application for a Warrant re: Case No. 2:22-MJ-03119. https://www.documentcloud.org/documents/24192891-search-warrant-for-google-account-for-push-notification-data. (Accessed on 06/01/2024).

[88] United States District Court for the District of Columbia. 2021. Application for a Warrant re: Case No. 21-sc-270. https://www.documentcloud.org/documents/24192911-6d68977d-f8ef-4080-9742-290cff8a6c28. (Accessed on 06/01/2024).

[89] Aleksandra Urman, Justin Chun-ting Ho, and Stefan Katz. 2021. Analyzing protest mobilization on Telegram: The case of 2019 anti-extradition bill movement in Hong Kong. *Plos one* 16, 10 (2021), e0256675.

[90] U.S. Congress. 1986. H.R.4952 - Electronic Communications Privacy Act of 1986 . https://www.congress.gov/bill/99th-congress/house-bill/4952. (Accessed on 10/10/2023).

[91] Viber. 2023. Privacy Notice for California Residents. https://web.archive.org/web/20230310001732/https://www.viber.com/en/terms/ccpa-privacy-rights/. (Accessed on 10/10/2023).

[92] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D Breaux, and Jianwei Niu. 2018. Guileak: Tracing privacy policy claims on user input data for android applications. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, New York, NY, USA, 37–47.

[93] Ian Warren, Andrew Meads, Satish Srirama, Thiranjith Weerasinghe, and Carlos Paniagua. 2014. Push notification mechanisms for pervasive smartphone applications. *IEEE Pervasive Computing* 13, 2 (2014), 61–71.

[94] Mark Wickham. 2018. Push Messaging. *Practical Android: 14 Complete Projects on Advanced Techniques and Approaches* (2018), 135–172.

[95] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. 2015. Android permissions remystified: A field study on contextual integrity. In *24th USENIX Security Symposium (USENIX Security 15)*. 499–514.

[96] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. 2017. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, New York, NY, USA, 1077–1093.

[97] Primal Wijesekera, Joel Reardon, Irwin Reyes, Lynn Tsai, Jung-Wei Chen, Nathan Good, David Wagner, Konstantin Beznosov, and Serge Egelman. 2018. Contextualizing privacy decisions for better prediction (and protection). In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.
[98] Wikipedia. 2023. Chicken Gun. https://en.wikipedia.org/wiki/Chicken_gun.
[99] Kim Wuyts, Laurens Sion, and Wouter Joosen. 2020. Linddun go: A lightweight approach to privacy threat modeling. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 302–309.
[100] Ron Wyden. 2023. Wyden Smartphone Push Notification Surveillance Letter. https://www.wyden.senate.gov/imo/media/doc/wyden_smartphone_push_notification_surveillance_letter.pdf. (Accessed on 01/01/2024).
[101] Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. 2022. Lalaine: Measuring and characterizing non-compliance of apple privacy labels at scale. *arXiv preprint arXiv:2206.06274* (2022).
[102] Zhi Xu and Sencun Zhu. 2012. Abusing Notification Services on Smartphones for Phishing and Spamming.. In *WOOT*. 1–11.
[103] Sebastian Zimmeck, Rafael Goldstein, and David Baraka. 2021. PrivacyFlash Pro: Automating Privacy Policy Generation for Mobile Apps.. In *NDSS*. Internet Society, Reston, VA, USA, 18 pages.
[104] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel R Reidenberg, N Cameron Russell, and Norman Sadeh. 2019. MAPS: Scaling privacy compliance analysis to a million apps. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2019, 3 (2019), 66–86.
[105] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman M Sadeh, Steven M Bellovin, and Joel R Reidenberg. 2017. Automated Analysis of Privacy Requirements for Mobile Apps.. In *NDSS*. Internet Society, Reston, VA, USA, 15 pages.

Data Type	Description
Device or other IDs	Identifiers that relate to an individual device, browser or app. For example, an IMEI number, MAC address, Widevine Device ID, Firebase installation ID, or advertising identifier.
User IDs	Identifiers that relate to an identifiable person. For example, an account ID, account number, or account name.
Name	How a user refers to themselves, such as their first or last name, or nickname.
Phone number	A user's phone number.
Messages	Any other types of messages. For example, instant messages or chat content.

Table 4: Google Play Store's data types applicable to our study. Note that Google refers to the 'Messages' data type as 'Other in-app messages.'

A DATA TYPES

Table 4 enumerates the data types that we searched for during our analysis of Android apps. Google defines and uses these data types to populate the information presented to users in the form of privacy labels in the app's listing on Google Play Store [35].

B CODE ANALYSIS WORKFLOW

We used this set of questions to analyze the source code of apps in our data set. These questions can also assist with data flow mapping, or in other words, tracing data contained in a push notification from its creation until the notification is displayed to the user.

- Does the app's `AndroidManifest.xml` register a service that `extends FirebaseMessagingService`?
- Locate the Java .java (or Kotlin .kt) source file corresponding to the registered service.
- Which FCM methods (e.g., `onMessageReceived()`, `onNewToken()`, etc.) does the service override?
- The `onMessageReceived()` method gets invoked when the client app receives an FCM push notification. Does the service override `onMessageReceived()` method?
- Data payload contained in an FCM push notification can be accessed by calling `remoteMessage.getData()`. Does the `onMessageReceived()` method invoke `getData()` on its argument of type `RemoteMessage`?
- Is there any indication that `RemoteMessage` contains sensitive data, based on the names of the keys or logging?
- Trace the code execution from the `onMessageReceived()` method until the message is displayed to the user.
- Does `RemoteMessage` get passed as a parameter to any function?
- What mechanisms (if any) are in place to ensure that notification contents do not get leaked to Google's FCM server?