# Sicherheit und Privatheit von Push-Notification-Services für mobile Apps

Chantal Bruns
Hochschule Bonn-Rhein-Sieg
chantal.bruns@smail.inf.h-brs.de

Issam Hbib
Hochschule Bonn-Rhein-Sieg
issam.hbib@smail.inf.h-brs.de

Robert Bargon
Hochschule Bonn-Rhein-Sieg
robert.bargon@smail.inf.h-brs.de

## ABSTRACT

TODO

## CCS CONCEPTS

• **Security and privacy → Domain-specific security and privacy architectures**.

## KEYWORDS

push notifications, encryption, cloud messaging services

## 1 INTRODUCTION

ACM's consolidated article template, introduced in 2017, provides a consistent LaTeX style for use across ACM publications, and incorporates accessibility and metadata-extraction functionality necessary for future Digital Library endeavors. Numerous ACM and SIG-specific LaTeX templates have been examined, and their unique features incorporated into this single new template.

If you are new to publishing with ACM, this document is a valuable guide to the process of preparing your work for publication. If you have published with ACM before, this document provides insight and instruction into more recent changes to the article template.

The "`acmart`" document class can be used to prepare articles for any ACM publication — conference or journal, and for any stage of publication, from review to final "camera-ready" copy, to the author's own version, with *very* few changes to the source.

## 2 GRUNDLAGEN VON PUSH-BENACHRICHTIGUNGEN

TODO

### 2.1 Was ist Firebase?

Firebase ist eine Entwicklungsplattform, mit welcher es möglich ist Apps (mobil und webbasiert) in der Google Cloud Plattform zu entwickeln. Es bietet durch verschiedene Services, den Entwicklern die Möglichkeit die App-Entwicklung zu beschleunigen und die Stabilität und Leistung zu überwachen. Darunter auch die Push-Notifications, welche im Folgenden genauer erläutert werden.

### 2.2 Typischer Push-Notification-Flow

Um die Funktionalität der Push-Benachrichtigungen herzustellen, werden die SDK-APIs der mobilen Plattform z.B. Apple iOS oder Android, verwendet, um sich damit bei einem Cloud-Messaging-Anbieter registrieren zu können. Dazu gehören Apple Push Notification Service (APNs) für iOS und Firebase Cloud Messaging (FCM) für Android. Zuerst sendet der Cloud-Messaging-Anbieter ein Push-Token, welches wie eine Adresse funktioniert, mit dem es möglich ist das Gerät und die mobile App als Empfänger der Nachricht zu identifizieren. Daraufhin wird das Push-Token von der mobilen App an den Server einer Customer-Engagement-Plattform gesendet, denn das Push-Token muss dem Server bekannt sein, um eine Push-Benachrichtigung an den Empfänger senden zu können. Die Sicherheit, dass die personalisierten Inhalte nur vom beabsichtigten Benutzer empfangen werden, ist durch die Eins-zu-Eins Zuordnung zwischen dem Push-Token und der Kennung des Benutzers vom System gewährleistet. Der Remote-Server sammelt die Nutzerdaten und Interaktionen und gibt den App-Entwicklern die Möglichkeit z.B. den Nachrichteninhalt so zu personalisieren, dass er auf die Nutzungsmuster und -attribute der Kunden angepasst ist. Die personalisierte Nachricht wird an das Push-Token gebunden und vom Remote-Server an den Cloud-Messaging-Dienst der mobilen Plattform gesendet. Der Nachrichteninhalt wird dann vom Cloud-Messaging-Dienst an das richtige Gerät und die mobile App gesendet und dann anschließend als Benachrichtigung angezeigt.

### 2.3 FCM für Android

Um Firebase Cloud Messaging-basierte Push-Benachrichtigungen in einer Android-App zu implementieren, werden eine mobile App, eine Verbindung zum FCM-Server und ein Push-Server eines Drittanbieters benötigt. Zuerst muss Firebase zu dem Android-App-Projekt hinzugefügt werden. Danach ist es erforderlich sich bei FCM zu registrieren, um das FCM-Registrierungstoken zu erhalten. Um dann anschließend eine Verbindung mit den FCM-Servern herzustellen, muss das FCM-Token in der Android-App konfiguriert werden. Nun wird ein Push-Server eines Drittanbieters benötigt, damit Benachrichtigungen von der mobilen App an den FCM-Server gesendet werden können. Hierfür wird oftmals der Google Play Store verwendet, der die Einrichtung mit den Google Play-Diensten APK erfordert. Es können aber auch Push-Benachrichtigungsserver eines Drittanbieters verwendet werden.

### 2.4 Push-Notifications für Apple

Für iOS funktionieren Push-Notifications ähnlich wie bei Android mit dem Unterschied, dass die App mit dem Apple Push Notification Service kommuniziert. Hierfür wird ein Apple Push Notification Authentication Key für den Apple Developer Account benötigt. Zuerst fordert iOS ein Gerätetoken vom Apple Push Notification Service (APNs) an. Danach empfängt die iOS-App das Token, mit welchem Push-Benachrichtigungen an die identifizierte Anwendung gesendet werden. Zum Schluss sendet der Server einer Customer-Engagement-Plattform die Benachrichtigung an den APNs und dann identifiziert der APNs das iOS-Gerät und die iOS-App, an die die Benachrichtigung gesendet werden soll.

### 2.5 Grundlagen zu unsicheren Push-Notifications

Firebase selbst bietet keine End-zu-End Verschlüsselung an, was die Übertragung der Push-Benachrichtigungen unsicher macht, da diese über Firebase unverschlüsselt übertragen werden und somit von Firebase ausgelesen und ausgewertet werden können. Stattdessen schlägt Firebase die Nutzung von Bibliotheken von Drittanbietern vor, unter anderem Capillary. Apple hingegen bietet das UNNotificationServiceExtension Objekt an, womit verschlüsselte Push-Notifications von der App entschlüsselt werden können. Dabei muss der versendete Notification Payload den Json Key „mutable-content“ mit dem Wert 1 beinhalten, damit das UNNotificationServiceExtension Objekt die verschlüsselte Notification entschlüsselt, bevor sie im Gerät angezeigt wird.

## 3 SICHERE PUSH-NOTIFICATIONS

TODO

### 3.1 Grundlagen von sicheren Push Notifications

Wenn eine Benachrichtigung eines App-Anbieters über einen Messaging Dienst wie FCM zum Endgerät gelangen soll, ist die Kommunikation zwischen Entwickler und Messaging Dienst, sowie die Kommunikation zwischen Messaging Dienst und Endgerät durch TLS verschlüsselt [1][2], jedoch wird der eigentliche Inhalt der Benachrichtigungen in Klartext übermittelt [1][2]. So handelt es sich dabei nicht um eine Ende-zu-Ende Verschlüsselung und der Inhalt der Nachricht kann von beispielsweise Google mitgelesen werden.

Bedenklich ist die Tatsache, dass alle Benachrichtigung recht einfach einer Person zugeordnet werden können und die Anbieter dieser Messaging Dienste meist US-amerikanische Firmen sind, welche beispielsweise Daten an Behörden weiterreichen müssten, falls gefordert [3].

Um den Inhalt von Push-Benachrichtigungen vor den Messaging Diensten zu schützen, muss der App-Anbieter sich also selbst um eine Ende-zu-Ende-Verschlüsselung kümmern.

Dies kann besonders wichtig sein, wenn es sich um Benachrichtigungen handelt, welche sensible Personen Informationen enthalten. Wenn es sich um Benachrichtigungen von Instant Messaging Diensten handelt, oder vielleicht sensiblere Daten von Banking Apps oder vergleichbarem, ist eine Verschlüsselung der Inhalte besonders erstrebenswert.

Eine solche Ende-zu-Ende-Verschlüsselung lässt sich über eine asymmetrische Verschlüsselung recht gut realisieren, wobei das Endgerät das Schlüsselpaar erzeugt und den Öffentlichen Schlüssel an den Entwickler schickt und dabei den Messaging Dienst außen vor lässt. Dabei muss man nicht die Funktionsweise des Messaging Dienstes einschränken und kann vorhandene Technologien weiter nutzen.

### 3.2 Beispiel eines sicheren Notification Flow

1. Der Anwender erzeugt auf seinem Endgerät automatisiert ein Schlüsselpaar, bestehend aus einem Öffentlichen und einem Privaten Schlüssel.

2. Der Öffentliche Schlüssel wird dem Entwickler nun über einen direkten Kommunikationsweg übermittelt.

3. Der Entwickler speichert nun diesen Schlüssel und verknüpft ihn Lokal mit dem Messaging Dienst Identifier für das spezifische Endgerät. Nun ist er in der Lage mit dem Öffentlichen Schlüssel seine Ausgehende Benachrichtigen zu verschlüsseln.

4. Die Verschlüsselte Benachrichtigung wird nun dem Messaging Dienst übermittelt, welcher diese, für ihn unlesbare Nachricht, an das Endgerät übermittelt.

5. Das Endgerät kann bei Erhalt der Benachrichtigung, durch seinen lokal gespeicherten Privaten Schlüssel den Inhalt entschlüsseln und die Benachrichtigung kann dem Nutzer angezeigt werden.

### 3.3 Umsetzung in Open Source Projekten

Capillary ist eine Bibliothek von Google, welche Drittanbieter, eine End-zu-End Verschlüsselung über Firebase Cloud Messaging ermöglicht. Unter anderem bietet sie die Schlüsselgeneration als auch Registrierung der erstellten asymmetrischen Schlüssel mit dem Developer Application Server, der die Benachrichtigungen verschlüsselt über Firebase an die App mit Capillary versendet. Zudem bietet Capillary die Entschlüsselung und Verschlüsselung der Notification clientseitig in der App als auch serverseitig an.

Capillary bietet noch weitere Funktionen, wie zum Beispiel, dass ein gestohlenes Endgerät die Möglichkeit der Entschlüsselung noch eingehender Benachrichtigungen entzogen werden kann [2].

Die Popularität dieser Bibliothek scheint, gemessen anhand der ungefähr 450 Github Stars Stand Dezember 2021, recht eingeschränkt. Der letzte Commit war im Dezember 2018, womit das Projekt recht verlassen ausschaut.

Eine weitere Möglichkeit verschlüsselte Notifications zu senden, besteht über das Extensible Messaging and Presence Protocol (XMPP). XMPP selbst ist ein Standard der Internet Engineering Taswk Force (IETF), welcher Technologien wie instant messaging, Push Notifications, voice und video calls über XML ermöglicht. Dieser Standard definiert zudem die End-zu-End Verschlüsselung als auch Signierung von versendeten Objekten. Dies kann verwendet werden, um Push Notifications zu verschlüsseln.

## 4 PROTOTYP

Im Rahmen des Projektes sollte überprüft werden, wie oft unsichere Push Notifications in Apps benutzt wurden. Hierfür wurden nur Android Apps betrachtet. Das erste Analysekriterium achtet auf die Benutzung von „NotificationCompat“ mit Hilfe von einem Stringabgleich, welche zu den Android Schnittstellen gehört und das Senden von Push Nachrichten ermöglicht. Wenn diese vorhanden ist, wird davon ausgegangen, dass eine App Notifications abschicken kann und dann wird auf die nächsten beiden Analysekriterien geachtet. Die zweiten und dritten Analysekriterien untersuchen die Apps auch mit einem Stringabgleich nach den beiden Strings „Capillary“ und „XMPP“. Ist der String „Capillary“ vorhanden, so wird davon ausgegangen, dass die App Capillary unterstützt, da diese Bibliothek zum Zweck der End-zu-End Verschlüsselung von Push Notifications erstellt wurde. Ist der String „XMPP“ vorhanden, so kann nicht zwingend davon ausgegangen werden, dass eine End-zu-End Verschlüsselung unterstützt wird, da die Ende-zu-Ende Verschlüsselung im Standard nur als optionales Feature definiert wurde.

### 4.1 TODO

Zum Herunterladen der Apps wurde das Fork vom Python Projekt GooglePlayAPI von TheZ3ro benutzt, zudem musste das Requests Python Projekt auf die Version 2.20.0 herabgestuft werden, da eine höhere Version inkompatibel ist. Als weiteres wurde ein eigenes Python Skript geschrieben, welches die GooglePlayAPI Bibliothek importiert und für den Login benutzt. Bei dem Login wird ein neues Gerät mit der Sprachumgebung, die Zeitzone und einem Gerätecodenamen mit dem benutzten Account initialisiert. Im Falle der Tests waren dies die Werte en_US für ein Gerät im US-Amerikanischen Raum, die Zeitzone UTC und der Gerätecodename hero2lte, welches für das Samsung Galaxy S7 Edge steht.

### 4.2 TODO

Nach dem erfolgreichen Login wird ein Logger für das LogFile als auch entsprechende Counter für die Anzahl tatsächlich überprüfter Apps, da sich manche Apps nicht herunterladen lassen und die Anzahl von Apps mit den Strings „Capillary“ und „XMPP“. Danach wird ein CSV Reader benutzt, um eine Datei auszulesen, die aus Application IDs der zu testenden Apps getrennt mit Zeilenumbrüchen besteht. Hierbei wurden mit einem Javascript Skript und nachträglicher manuellen Anpassung aus der Seite „androidrank.org“ die Application IDs der Top 500 Apps der Bereiche „Health_And_Fitness“ und „Medical“ und in entsprechende zwei Dateien zusammengefasst, welche dem Skript zugefügt wurden.

### 4.3 TODO

Sobald die erste Application ID gelesen wurde, versucht das Skript die App in einen temporären Ordner herunterzuladen. Schlägt dies fehl, wird die nächste App iteriert. Wurde die App erfolgreich heruntergeladen, wird diese zunächst mit dem unzip Befehl entpackt, dann wird daraufhin der Inhalt der entpackten App mit dem grep Befehl nach den oben genannten Kriterien überprüft und zuletzt die entsprechenden Counter erhöht, sowie die Application ID mit dem gefundenen Wert in die LogDatei geschrieben, falls die App Capillary und/oder XMPP benutzt.

## ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

## REFERENCES

## A RESEARCH METHODS

### A.1 Part One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.

### A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

## B ONLINE RESOURCES

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.