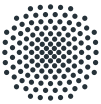


Universität Stuttgart

Konzepte für Datensicherheit und Datenschutz in mobilen Anwendungen

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Christoph Stach

aus Backnang

Hauptberichter: Prof. Dr. Bernhard Mitschang

Mitberichter: Prof. Dr. Ralf Küsters

Tag der mündlichen Prüfung: 29. November 2017

Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftware

2017

Mein Werk ist verrichtet,
die Grenze gefallen,
Unwissenheit ein
gebrochener Fluch.

— Saltatio Mortis, Prometheus

Inhaltsverzeichnis

Abkürzungsverzeichnis

A

ABE *Attribute-Based Encryption*
ACL *Access Control List*
ADT *Android Developer Tools*
AES *Advanced Encryption Standard*
AIDL *Android Interface Definition Language*
AIS *Application Information Set*
API *Application Programming Interface*
APK *Android Application Package*
ASLR *Address Space Layout Randomization*
AWS *Amazon Web Services*

B

BE *Broteinheit*
BPMN *Business Process Model and Notation*
BYOD *Bring Your Own Device*

C

CC *Candy Castle*
CEP *Complex Event Processing*

COPD *Chronic Obstructive Pulmonary Disease*
CP-ABE *Ciphertext-Policy ABE*

D

DAC *Discretionary Access Control*
DEX *Dalvik Executable*
DoD *Department of Defense*
DRM *Digital Rights Management*

E

EAS *Exchange ActiveSync*
EIS *Editor for Information Sets*
ERC *Elevated Rights Chamber*

F

FDE *Full-disk Encryption*

G

GCHQ *Government Communications Headquarters*
GPS *Global Positioning System*
GSM *Global System for Mobile Communications*
GUI *Graphical User Interface*

I

IoT *Internet of Things*
IPC *Inter-Process Communication*

J

JAR *Java Archive*
JSON *JavaScript Object Notation*

K

KP-ABE *Key-Policy ABE*

L

LPC *Least Privileged Chamber*

M

MAC *Mandatory Access Control*
MMD *Mobile Manufacturing Dashboard*
MMS *Multimedia Messaging Service*

N

NSA *National Security Agency*

O

OEM *Original Equipment Manufacturer*

P

PbD *Privacy by Design*
PDC *PMP-Datencontainer*
PDE *Plausibly Deniable Encryption*
PID *Process Identifier*
PIN *Personal Identification Number*
PMP *Privacy Management Platform*
PPM *Privacy Policy Model*
PSS *Proportional Set Size*
PUA *Potentially Unwanted Application*

R

RBAC *Role Based Access Control*
RFID *Radio-Frequency Identification*

RGIS *Resource Group Information Set*
ROM *Read-Only Memory*

S

SCC *Secure Candy Castle*
SDC *Secure Data Container*
SDC+ *Secure Data Container Plus*
SDK *Software Development Kit*
SMS *Short Message Service*
SQL *Structured Query Language*
SRC *Standard Rights Chamber*

T

TCB *Trusted Computing Base*
TPM *Trusted Platform Module*

U

UEFI *Unified Extensible Firmware Interface*
UGC *User-Generated Content*
UID *User Identifier*
UML *Unified Modeling Language*
URI *Uniform Resource Identifier*

W

WLAN *Wireless Local Area Network*

X

XACML *eXtensible Access Control Markup Language*
XML *Extensible Markup Language*
XN *Execute Never*
XSD *XML Schema Definition*

Kurzfassung

Smart Devices und insbesondere Smartphones nehmen eine immer wichtigere Rolle in unserem Leben ein. Aufgrund einer kontinuierlich anwachsenden Akkulaufzeit können diese Geräte nahezu ununterbrochen mitgeführt und genutzt werden (*always on*). Zusätzlich sorgen stetig günstiger werdende Mobilfunktarife und ansteigende Datenraten dafür, dass den Nutzern mit diesen Geräten eine immerwährende Verbindung zum Internet zur Verfügung steht (*always connected*). Smart Devices sind dadurch nicht mehr reine Kommunikationsmittel sondern ebenfalls Informationsquellen. Darüber hinaus gibt es eine Vielzahl an Anwendungen von Drittanbietern für diese Geräte. Dank der darin verbauten Sensoren, können darauf beispielsweise ortsbasierte Anwendungen, Gesundheitsanwendungen oder Anwendungen für die Industrie 4.0 ausgeführt werden, um nur einige zu nennen.

Solche Anwendungen stellen allerdings nicht nur ein großes Nutzen-, sondern zu gleich ein immenses Gefahrenpotential dar. Über die Sensoren können die unterschiedlichsten Kontextdaten erfasst und relativ präzise Rückschlüsse auf den Nutzer[1] gezogen werden – ganz zu schweigen von den Daten die der Nutzer selbst auf dem Gerät speichert (z. B. Fotos, Kontaktdaten oder Termine). Diese Daten sind ein wesentlicher Grund dafür, dass es in Anwendungen für Smart Devices vergleichsweise viel Schadcode gibt, der Informationen über den

[1] Im Rahmen dieser Arbeit wird aus Gründen der besseren Lesbarkeit auf Beidnennung, Neutralisierung sowie Sichtbarmachung bewusst verzichtet. Das hier verwendete generische Maskulinum schließt daher stets sämtliche Geschlechter mit ein.

Nutzer sammelt und diese an den Hersteller der Anwendung schickt. Daher sollte bei diesen Geräten ein besonderes Augenmerk auf die Datensicherheit (*Information Security*) und insbesondere auf den Datenschutz (*Data Privacy*) gelegt werden.

Betrachtet man allerdings die bestehenden Datensicherheits- und Datenschutzkomponenten in den aktuell vorherrschenden mobilen Plattformen (Apple iOS, Microsoft Windows Phone und Google Android), so fällt auf, dass keine der Plattformen die speziellen Anforderungen an ein mobiles Datensicherheits- und Datenschutzsystem zufriedenstellend erfüllt. Während iOS zwar vielschichtige Kontrollen der verfügbaren Anwendungen einführt und nur solche zur Veröffentlichung freigibt, die den Sicherheitsansprüchen von Apple genügen, wird der Nutzer dabei vollständig ausgeschlossen. Er muss blind darauf vertrauen, dass die Kontrollen von Apple gründlich und in seinem Interesse sind. Individuelle Datenschutzeinstellungen, z. B. um einer Anwendung zu verbieten auf die Werte eines speziellen Sensors zuzugreifen, sind dabei nicht vorgesehen.

Vollständig anders handhabt Android den Schutz privater Daten. Jede Anwendung kann grundsätzlich auf alle Daten Zugriff bekommen, wenn sie dies dem Nutzer bei der Installation bekanntgibt. Um die Anwendung allerdings installieren zu können, muss er allen Zugriffsanfragen entsprechen. Somit liegt die Verantwortung über die Daten vollständig in den Händen des Nutzers. Allerdings sind die gegebenen Einstellungsmöglichkeiten zur Zugriffsbeschränkung nicht ausreichend feingranular und flexibel, um dem Datenschutz Herr zu werden.

Aufgrund der erheblichen Schwächen dieser Datenschutzsysteme von mobilen Plattformen, sind Sicherheitserweiterungen ein immerwährender Gegenstand der Forschung. Dabei gibt es zwei unterschiedliche Implementierungsstrategien: Entweder wird die Erweiterung direkt in die mobile Plattform eingebracht oder es werden alle Anwendungen umgeschrieben, so dass diese mit der Erweiterung zusammenarbeiten. Die Wahl der Implementierungsstrategie hat allerdings weitreichende Folgen. Die Modifikation der mobilen Plattform führt dazu, dass der Nutzer die Plattform vollständig neu bei sich aufsetzen muss. Dies schreckt insbesondere Laien von der Verwendung ab und der Garantieanspruch geht durch die Manipulation verloren. Die Manipulation der Anwendungen stellt hingegen einen Verstoß gegen geltendes Urheberrecht

dar und kann sowohl für den Anbieter als auch für den Nutzer rechtliche Konsequenzen nach sich ziehen.

Aus diesem Grund steht im Zentrum der vorliegende Arbeit die Konzeption und Umsetzung neuartiger Datensicherheits- und Datenschutzkonzepte für mobile Anwendungen. Hierfür werden die folgenden fünf Forschungsbeiträge erbracht:

(**FB$_1$**) Bestehende Datensicherheits- und Datenschutzkonzepte werden analysiert, um deren Schwachstellen zu identifizieren.

(**FB$_2$**) Ein kontextsensitives Berechtigungsmodell wird erstellt.

(**FB$_3$**) Das Berechtigungsmodell wird in einem flexiblen Datenschutzsystem konzeptionell eingebettet und anschließend implementiert.

(**FB$_4$**) Das Datenschutzsystem wird zu einem holistischen Sicherheitssystem erweitert.

(**FB$_5$**) Das daraus entstandene holistische Sicherheitssystem wird evaluiert.

Um die Forschungsziele zu erreichen, wird mit dem *Privacy Policy Model (PPM)* ein gänzlich neues Modell zur Formulierung von Berechtigungsregeln eingeführt. Das *PPM* folgt dem *Least-Privilege*-Prinzip, d. h. es ist dadurch möglich, Anwendungen sehr feingranular ein Mindestmaß an Berechtigungen zu geben. Zu diesem Zweck teilt das *PPM* jede Anwendung in kleinere Funktionseinheiten auf und vergibt jeder Einheit genau die Berechtigungen, die für deren Ausführung notwendig sind. Dies ermöglicht dem Nutzer je nach Bedarf, einzelne Funktionseinheiten zu deaktivieren, um dadurch die Zugriffsrechte der Anwendung einzuschränken. Zusätzlich kann der Nutzer auch die Genauigkeit der Daten, die der Anwendung zur Verfügung gestellt werden, reduzieren. Das *PPM* ist darüber hinaus kontextsensitiv, d. h. jeder darin beschriebenen Regel kann ein Kontext zugeordnet werden, in dem sie gelten soll.

Das *PPM* wird in der *Privacy Management Platform (PMP)* implementiert. Die *PMP* ist ein Berechtigungssystem, das nicht nur für die Einhaltung der Datenschutzrichtlinien sorgt, sondern auch einige der Schutzziele der Datensicherheit erfüllt. Zu diesem Zweck erweitert sie den Installationsprozess von Anwendungen dahingehend, dass sie dabei analysiert, welche Funktionseinheiten in der Anwendung definiert sind und welche Berechtigungen von diesen

benötigt werden. Basierend auf dieser Analyse kann der Nutzer eine initiale Konfiguration der Berechtigungsregeln vornehmen. Wenn eine Anwendung auf Daten zugreifen will, dann stellt sie die Anfrage an die *PMP*, die prüft, ob der Datenzugriff den Berechtigungsregeln entspricht. Ist dies der Fall, führen Erweiterungen der *PMP*, die sogenannten *Resources*, den Datenzugriff durch und leiten die Daten an die anfragende Anwendung zurück. Besitzt die Anwendung nicht die benötigten Berechtigungen, so informiert die *PMP* den Nutzer darüber und er kann auf Wunsch zur Laufzeit Änderungen an den Berechtigungsregeln durchführen. Auf diese Weise ist sichergestellt, dass eine Anwendung aufgrund fehlender Berechtigungen nicht abstürzt, wie es bei vergleichbaren Ansätzen der Fall ist. Es werden lediglich die betroffenen Funktionseinheiten deaktiviert, während der Rest der Anwendung davon unberührt bleibt. Eine wesentliche Eigenschaft der *PMP* ist, dass diese erweiterbar ist, d. h. es können jederzeit weitere *Resources* installiert werden, um zusätzliche Datenquellen zu unterstützen.

Für die *PMP* werden mehrere Implementierungsstrategien diskutiert und deren Vor- und Nachteile gegeneinander abgewogen. Hierzu zählen u. a. die Implementierung als eigenständige Anwendung, die Implementierung als Erweiterung des zugrundeliegenden Betriebssystems oder die Implementierung mit einem Konverter für Anwendungen von Drittanbietern. Da keine dieser Implementierungsstrategien vollständig überzeugt, wird mit dem *PMP-Gatekeeper* eine neuartige Implementierungsstrategie speziell für die *PMP* eingeführt. Der *PMP-Gatekeeper* ist eine Analysekomponente, die Anwendungen zuverlässig in Gruppen einteilen kann, für die jeweils unterschiedliche Berechtigungssystem zum Einsatz kommen können.

Um neben den Datenschutz auch die Datensicherheit gewährleisten zu können, wird die *PMP* um den *Secure Data Container (SDC)* erweitert. Mit dem *SDC* können sensible Daten sicher gespeichert und zwischen Anwendungen ausgetauscht werden. Hierfür bietet der *SDC* ein generisches Datenbankschema, das sich auf unterschiedliche Anwendungsfälle anpassen lässt. Die im *SDC* gespeicherten Daten sind dauerhaft verschlüsselt und nur der *SDC* besitzt den Schlüssel, so dass sie gegen Angriffe von außen geschützt sind. Dabei bietet der *SDC* unterschiedliche Verschlüsselungsverfahren an und der Nutzer kann selbst für jeden Anwendungsfall einen geeigneten Kompromiss aus Verschlüs-

selungsstärke und -geschwindigkeit treffen. Da der *SDC* als Erweiterung für die *PMP* konzipiert ist, erfolgt der Datenzugriff stets über die *PMP*. Der Datenspeicher ist daher nicht einer bestimmten Anwendung zugeordnet, sondern kann prinzipiell von jeder Anwendung genutzt werden. Im *SDC* werden in Metadaten hinterlegt, welche Anwendungen Zugriff auf welche Daten erhalten sollen. Die Granularität dieser Zugriffsrechte kann dabei sogar bis auf die Tupelebene gehen, wenn dies vom Nutzer gewünscht ist. Der *SDC* eignet sich allerdings nicht nur als sicherer Datenspeicher, sondern kann auch dafür genutzt werden, um Daten zuverlässig zu löschen. Dies ist insbesondere auf Flash-Speichern, wie sie in Smart Devices zum Einsatz kommen, nicht einfach. Im Falle des *SDCs* müssen allerdings nicht die Daten selbst gelöscht werden, sondern nur der Schlüssel zu diesen, da sie ohne diesen Schlüssel für einen Angreifer nicht lesbar und damit wertlos sind. In einer Erweiterung des *SDCs*, dem *Secure Data Container Plus (SDC+)*, werden Nutzerkonten zusammen mit einem sicheren Mechanismus zur Eingabe von Anmeldeinformationen einführt. Durch die Nutzerkonten ist es möglich, Zugriffsrechte an eine Kombination aus Nutzer und Anwendung zu binden. Dadurch hat jeder Nutzer nur Zugriff auf seine eigenen Daten. Darüber hinaus können auf diese Weise Daten nicht nur zwischen Anwendungen sondern auch zwischen Nutzern ausgetauscht werden.

Die Anwendbarkeit der *PMP* und des *SDCs* wird an Praxisbeispielen aus vier unterschiedlichen Domänen demonstriert. Diese Anwendungsfälle sind ortsbasierte Anwendungen, Gesundheitsanwendungen, Anwendungen in der Industrie 4.0 sowie Anwendungen für das Internet der Dinge. Bei dieser Analyse zeigt sich, dass die Kombination aus *PMP* und *SDC* nicht nur sämtliche Schutzziele, die im Rahmen der vorliegenden Arbeit relevant sind und sich am ISO-Standard ISO/IEC 27000:2009 orientieren, erfüllt, sondern darüber hinaus sehr performant ist. So beschleunigt sich durch die vereinfachte gemeinsame Datenhaltung im *SDC* die Laufzeit von Anwendungen sowie die von diesen verursachte CPU-Auslastung maßgeblich. Dadurch kann durch die Verwendung der *PMP* und des *SDCs* der Akkuverbrauch von Anwendungen halbiert werden.

Damit die *PMP* ihre volle Wirkung entfalten kann, sind Anpassungen seitens der Entwickler am Quellcode der Anwendungen nötig. Zur Unterstützung

der Entwickler wurden im Rahmen der vorliegenden Arbeit auch Werkzeuge und Konzepte entwickelt, die die Umstellung von bestehenden Anwendungen auf die *PMP* erleichtern. Zum einen wird eine Eclipse-Erweiterung zur Erstellung der von der *PMP* benötigten Metadaten vorgestellt. Über Eingabemasken kann der Entwickler dadurch die Metadaten semi-automatisch erstellen. Zum anderen wird mit *Gamework* ein Framework für autoadaptive Anwendungen vorgestellt. Dieses Framework erleichtert die Entwicklung von Anwendungen für Smart Devices, indem darin häufig benötigte Funktionen in vorkonfigurierten Bausteinen vorkonfiguriert zur Verfügung gestellt werden. Darüber hinaus wird *Gamework* konzeptionell um eine Feedback-Schleife erweitert, durch die Anwendungen automatisch an die Nutzeranforderungen angepasst werden. Im Rahmen der *PMP* kann diese Feedback-Schleife dafür genutzt werden, dass Anwendungen automatisch an die Berechtigungsvorgaben des Nutzers angepasst werden, um ihm einen möglichst großen Service bieten zu können.

Mithilfe dieser Komponenten kann gezeigt werden, dass die aufgestellten Forschungsziele vollständig erreicht werden. Sämtliche Komponenten wurden prototypisch für Android implementiert. Die zugrundeliegenden Sicherheitskonzepte lassen sich jedoch leicht auf andere IT-Umgebungen übertragen, die mit einer Vielzahl an sensiblen Daten umgehen, wie beispielsweise die *Facebook Platform*, *Chromium OS* oder *AWS*. Summa summarum lässt sich festhalten, dass die in der vorliegenden Arbeit eingeführten Konzepte zum Management der Datensicherheit und des Datenschutzes alle aufgestellten Schutzziele erfüllen.

Abstract

Smart Devices and particularly Smartphones are becoming more and more relevant in our everyday lives. Due to an increasing battery service life, these devices became a constant companion and can be used anywhere at anytime (*always on*). In addition, cheap cell phone tariff plans and growing data rates ensure that users have an stable connection to the Internet via these devices (*always connected*). Smart Devices are no longer just a communication tool, but rather an information source. Furthermore, there is a variety of third-party applications for these devices. Due to the built-in sensors, Smart Devices are an enabler for location-based applications, healthcare applications, or Industry 4.0 applications, just to name a few.

However, such applications are not only beneficial for their users, but also possess an immense threat potential. The built-in sensors can be used to capture almost any kind of context data and to draw relatively precise conclusions about a user—not to mention the huge amount of user data stored on the device (e. g., photos, contact information, or appointments). This data is a key reason for a comparatively high number of malicious code in applications for Smart Devices. This malicious software collects information about the user and sends it to the application developer. This is why users of Smart Devices should be fully aware of the importance of information security measures and especially data privacy measures.

Looking at existing information security and data privacy mechanisms in currently prevailing mobile application platforms (Apple iOS, Microsoft Win-

dows Phone, and Google Android), it is striking that none of them meets all specific requirements of a mobile data security system adequately. For instance, iOS introduces a multi-layered vetting mechanism which inspects any available application. Only applications that stand up to Apple's scrutiny are accredited for publication. However, this procedure expels the user completely. S/he has to trust blindly that reviews are thorough and that Apple's vetting policy is on his or her behalf. Individual privacy settings, e. g., in order to prohibit an application from accessing the values of a specific sensor, are not intended.

Android opts for a completely different strategy for protecting private data. Every application is able to access any kind of data as long as the application notifies the user about the required permissions during the installation process. While the responsibility to protect his or her data is entirely in the hands of the user, the given configuration capabilities for restricting the access to the data are neither sufficiently fine-grained nor flexible to ensure the protection of privacy. As a consequence, the user has to grant each of the application's permissions, in order to install the application.

Due to the considerable deficiencies of the information security and data privacy mechanisms, a lot of research is done concerning security extensions for mobile platforms. There are two distinct implementation strategies for such extensions: The extension is either hooked directly into the mobile platform or all applications are automatically rewritten so that they collaborate with the extension. However, the implementation strategy has far-reaching consequences on the usability of the extension. On the one hand, a manipulation of the mobile platform itself entails that the user has to rebuild and reinstall the whole system manually. This deters many users from installing such an extension, especially since they lose their warranty claim in the process. On the other hand, the manipulation of applications violates applicable copyright law and has therefore legal consequences for both the extension developer and for the user of such an extension.

For these very reasons, the focus of the thesis in hand lies on the design and implementation of novel information security and data privacy concepts for mobile applications. Thereby, it provides the following five research contributions:

(**FB$_1$**) Existing information security and data privacy concepts are analyzed in order to identify vulnerabilities.

(**FB$_2$**) A context-sensitive privacy policy model is created which solves these vulnerabilities.

(**FB$_3$**) The privacy policy model is conceptually embedded in a flexible data privacy system, which is subsequently implemented.

(**FB$_4$**) The data privacy system is extended to a holistic data security system[1].

(**FB$_5$**) The resulting holistic data security system is evaluated regarding the protective goals and its performance.

As a first step, the *Privacy Policy Model (PPM)*, a completely new model for formulating privacy policy rules, is introduced. The *PPM* follows the principle of *least privilege*. That is, it enables users to grant applications requested permissions on a very fine-grained level. To this end, the *PPM* divides every application into smaller functional units and assigns only those permissions to each unit which are absolutely necessary for its execution. Thereby, the user is able to activate and deactivate any functional unit according to his or her needs in order to restrict relevant permissions of the application. In addition, the user is also able to reduce the accuracy of the data provided to the application. Furthermore, the *PPM* is context-sensitive, i. e., each privacy policy rule can be assigned to a context in which it should be applied.

The *PPM* is implemented in the *Privacy Management Platform (PMP)*. The *PMP* is primarily a data privacy system. However, it also meets several objectives concerning information security. To this end, the *PMP* alters the application installation process by adding an analysis phase. In this phase, the *PMP* identifies the function units that are defined within the application and the permissions that are required by them. Based on this analysis, the user can make an initial configuration of the privacy policy rules. During its execution, an application has to access any kind of data via the *PMP*. The *PMP* checks whether the data access complies with the privacy policy rules. If this is the case, *PMP* extensions, so-called *Resources*, perform the data access and return

[1] Data privacy is the practice of determining what data can be shared with third parties. Information security is the practice of preventing unauthorized access to and processing of data. A holistic data security system accomplishes both of these goals.

the data to the requesting application. If the application violates the privacy policy, the *PMP* informs the user and s/he can change the privacy policy rules at runtime. This procedure ensures that an application does not crash due to missing permissions, as it is the case for related approaches. The *PMP* deactivates only affected functional units, while the rest of the application remains unaffected. An essential characteristic of the *PMP* is that it is extensible. That is, additional *Resources* can be installed at any time in order to support additional data sources.

Several implementation strategies for the *PMP* are discussed and their pros and cons are highlighted. These strategies include, among others, the implementation as a stand-alone application, the implementation as an extension to the underlying operating system, or the implementation using a tool rewriting the bytecode of third-party applications. Since none of these implementation strategies are completely convincing, the *PMP-Gatekeeper* is introduced as a novel implementation strategy specifically for the *PMP*. The *PMP-Gatekeeper* is an analysis component that classifies applications into domains for which different data privacy systems can be used.

In order to ensure not only data privacy, but also information security, the *PMP* is extended by the *Secure Data Container (SDC)*. With the *SDC*, sensitive data can be securely stored and shared among applications. For this purpose, the *SDC* offers a generic database schema, which can be tailored to various application scenarios. The data stored in this data container is permanently encrypted and only the *SDC* knows the key. Thereby, the data is protected against illegal access attempts. The *SDC* offers various encryption methods. So, the user can find a suitable compromise between encryption strength and processing speed for each application. As the *SDC* is designed as an extension to the *PMP*, the data is always accessed via the *PMP*. This way, the *SDC* is not assigned to a particular application, but can be used by every application. The *SDC* manages additional metadata that defines which applications should have access to which data. The granularity of these access rights may even go down to the tuple level if this is requested by the user. However, the *SDC* is not only a secure data storage, but can also be used to erase data reliably. As flash memory is used in Smart Devices, this is not an easy task. To solve this, the *SDC* does not delete the data itself, but only the key which is required to decrypt the

data. Without this key, the data are not readable and therefore worthless for an attacker. In an extension of the *SDC*, the so-called *Secure Data Container Plus (SDC+)*, user accounts are introduced in conjunction with a secure mechanism for entering login credentials. The user accounts enable to bind access rights to a combination of a user and an application. This means that each user has access to his own data, only. In addition, data can be exchanged not only between applications, but also between users via the *SDC+*.

The applicability of the *PMP* and the *SDC+* is demonstrated by applying them to real-world examples from four different domains. These use case scenarios include location-based applications, healthcare applications, Industry 4.0 applications, and applications for the Internet of Things. This analysis shows that the combination of the *PMP* and the *SDC* not only meets all objectives concerning data privacy and information security that are relevant to the work in hand, but that it is also highly performant. The privacy and security objectives are based on the ISO standard ISO/IEC 27000:2009. Due to the simplified sharing procedure for data stored in the *SDC*, the application runtime as well as the CPU utilization can be improved significantly. As a result of using the *PMP* and the *SDC*, the battery consumption caused by the execution of applications is reduced by approximately 50 %.

In order to take its full effect, the *PMP* requires the modification of an application's source code on behalf of the application developers. To support the developers in this process, tools and concepts are developed within the scope of the work in hand, which facilitate the adaptation of existing applications as required by the *PMP*. On the one hand, an Eclipse extension is introduced which simplifies the creation of *PMP* metadata. Via input masks, developers are able to create this metadata semi-automatically. On the other hand, *Gamework* is introduced, a framework for autoadaptive applications. This framework facilitates the development of applications for Smart Devices by providing pre-configured modules for functions which are often required by mobile applications. In addition, *Gamework* is conceptually extended by a feedback loop that automatically adapts applications according to user requirements. Within the scope of the *PMP*, this feedback loop can be used to automatically adapt applications to the user-defined privacy policy rules in order to provide the user with the greatest possible service quality.

All components were implemented prototypically for Android. The evaluation results show the success of our approach regarding its functionality, efficiency, and performance. In summary, it can be said that the data privacy and information security concepts for mobile applications introduced in the thesis in hand meet all the postulated objectives. Due to its modular expandability, the underlying security concepts can be easily transferred to other IT environments that have to deal with a variety of sensitive data, such as the *Facebook Platform*, *Chromium OS*, or *AWS*.

Danksagungen

Diese Dissertation entstand an der Abteilung Anwendersoftware des Instituts für Parallele und Verteilte Systeme (IPVS / AS) an der Universität Stuttgart.

Mein besonderer Dank gilt meinem Hauptberichter Prof. Dr. Bernhard Mitschang, der es mir ermöglichte, mehrere Jahre Teil seiner Abteilung zu sein und in diesem Rahmen meine Promotion voranzutreiben. Er hat mich bei meiner Arbeit angeleitet und stets mit großem Engagement unterstützt, ließ mir aber auch immer genügend Freiheiten zur Gestaltung meiner Forschung.

Ebenso möchte ich mich bei Prof. Dr. Ralf Küsters dafür bedanken, dass er den Mitbericht übernommen hat. Er hat durch seine Expertise im Bereich Datensicherheit meine Arbeit wesentlich bereichert.

Mein Dank gilt auch meinen Kollegen der Abteilung Anwendersoftware des Instituts für Parallele und Verteilte Systeme. Insbesondere seien an dieser Stelle Dr. Holger Schwarz, Michael Behringer, Dr. Nazario Cipriani, Ana Cristina Franco da Silva, Corinna Giebler, Dr. Christoph Gröger, Eva Hoos, Dr. Carlos Lübbe, Dr. Peter Reimann, Frank Steimle sowie Dr. Tim Waizenegger erwähnt, mit denen ich zahlreiche fachliche Diskussionen im Rahmen meines Promotionsvorhaben hatte, die in mehreren gemeinsamen Veröffentlichungen mündeten.

Besonders hervorheben möchte ich, dass Dr. Holger Schwarz, Michael Behringer, Dr. Peter Reimann, Bettina Stach, Frank Steimle sowie Dr. Stefan Zimmer durch Detaildiskussionen, zahlreiche Ratschläge und konstruktive Kritik die Qualität meiner Arbeit förderten.

Bei Dr. Christoph Fehling, Katharina Görlach, Dr. Florian Haag, Dr. André van Hoorn, Dr. Steffen Lohmann, Dr. Sascha Riexinger, Dr. Johannes Wettinger und Dr. Stefan Zimmer möchte ich mich für deren große Unterstützung bei der Gestaltung, Organisation und Durchführung gemeinsamer Lehrveranstaltungen bedanken.

Darüber hinaus gilt mein Dank Hermann Hörth, Heike Kniehl, Michael Matthiesen, Stefanie Palmer, Manfred Rasch, Ulrike Ritzmann, Annemarie Roesler, Katrin Schneider, Christine Schütz, Eva Strähle sowie Christine Well, die mir bei administrativen Fragen stets helfend zur Seite standen.

Ralf Aumüller, Martin Brodbeck, Bernd Schusser, Andreas Schaupp sowie Christine Reissner danke ich für ihre fortwährende technische Unterstützung.

Ich möchte mich auch bei den vielen von mir betreuten Studenten bedanken, die mit ihren Abschlussarbeiten und Implementierungen zum Erfolg meiner Forschung beigetragen haben.

Ein ganz spezielles ευχαριστώ πολύ geht an Iouliana Litou, nicht nur für die schöne Zeit auf der MDM '14 in Brisbane und der MDM '16 in Porto, sondern insbesondere für die Erkenntnis, dass deutsche Servietten und griechische Damenbinden zwar nahezu identisch klingen, man bei einem Konferenz-Dinner dennoch ziemlich genau darauf achten sollte, welches der beiden Produkte man seinem Gegenüber anbietet!

Meiner Familie und meinen Freunden danke ich für ihre fortwährende Unterstützung, ihr Verständnis und ihre Geduld auf meinem Weg zur Promotion.

Kapitel 1

Einleitung

> *These days, it's hard to imagine a life without mobile devices. In a time of ubiquitous smart phones, you have access to an entire world of ideas [...], and you always have something to occupy your attention [...]. The* smart *phone is aptly named.*
>
> — *Eric Schmidt und Jared Cohen [SC13a]*

Kaum treffender als in dem Zitat von Schmidt und Cohen [SC13a] lässt sich beschreiben, wie Smart Devices, insbesondere aber Smartphones, in der Bevölkerung wahrgenommen werden: Sie sind unsere ständigen Begleiter, da sie für uns eine immerwährende Verbindung zum Internet darstellen (*always connected*) und somit als Informationsquelle und Kommunikationsmittel in jeder Lebenslage genutzt werden können [WDSH12]. Da die Akkulaufzeit dieser Geräte kontinuierlich anwächst, können sie zunehmend länger aktiviert bleiben (*always on*) [EME+11]. Die vielleicht relevanteste Eigenschaft eines Smart Devices, die diesen Geräten endgültig zum Durchbruch verholfen hat, ist jedoch die Möglichkeit, darauf Anwendungen von Drittanbietern zu

installieren [Sag12]. Da sich zusätzlich die Anzahl der verbauten Sensoren, mit denen der Kontext des Nutzers[1] erfasst und in Anwendungen verwendet werden kann, stetig erhöht [Luc15], realisieren diese Geräte Mark Weisers Vision vom *Computer des 21. Jahrhunderts* [Wei91].

Durch diese neuen Verwendungsarten ergeben sich allerdings neue Gefahrenquellen für die Datensicherheit (*Information Security*) und insbesondere für den Datenschutz (*Data Privacy*) [Lea11], zumal der Wert unserer Daten beständig steigt [KR11]. So wird mit zunehmender Häufigkeit davon berichtet, dass Dritte auf die private Daten eines Nutzers, wie beispielsweise dessen Fotos, ohne seine Einwilligung Zugriff erlangen [Bil12]. In der vorliegenden Arbeit werden daher Datenschutzkonzepte für mobile Anwendungen (den sogenannten *Apps*) entwickelt und analysiert. Diese Konzepte werden im Rahmen der *Privacy Management Platform (PMP)* und deren Erweiterungen, insbesondere dem *Secure Data Container (SDC)*, implementiert und evaluiert.

Abschnitt 1.1 liefert zunächst eine detaillierte Motivation für diese Arbeit, bevor Abschnitt 1.2 das Verständnis der drei Begrifflichkeiten Datensicherheit (Abschnitt 1.2.1), Datenschutz (Abschnitt 1.2.2) und Vertrauen (Abschnitt 1.2.3) im Kontext der Arbeit abgrenzt. Anschließend definiert Abschnitt 1.3 Schutzziele, die erfüllt werden müssen, um die Datensicherheit und den Datenschutz von Apps gewährleisten zu können und somit das Vertrauen der Anwender in die Apps zu erhöhen. Die Forschungsbeiträge der vorliegenden Arbeit, mit denen diese Schutzziele erfüllt werden können, werden in Abschnitt 1.4 vorgestellt. Abschnitt 1.5 schließt dieses einleitende Kapitel mit einem Überblick über den Aufbau der Arbeit ab.

1.1 Motivation

Die heutige Gesellschaft durchlebt einen Wandel: Die Gewinnung, Speicherung, Verarbeitung und Nutzung von Informationen und Wissen spielt eine immer entscheidendere Rolle. Von dieser sogenannten *Informationsrevolution* sind nicht nur wirtschaftliche Organisationen betroffen, sondern im gleichen

[1] Im Rahmen dieser Arbeit wird aus Gründen der besseren Lesbarkeit auf Beidnennung, Neutralisierung sowie Sichtbarmachung bewusst verzichtet. Das hier verwendete generische Maskulinum schließt daher stets sämtliche Geschlechter mit ein.

Maße auch Privatpersonen [Pil00]. Die Geschwindigkeit, mit der dieser Wandel vonstatten geht, ist eng mit den Fortschritten in den Informations- und Telekommunikationstechnologien verknüpft [SPJW12]. Besonders im privaten Sektor müssen hier primär Smartphones als Enabler genannt werden [TGHI13]. Im Rahmen dieser Arbeit wird ein Smartphone gemäß Beal [Bea08] von einem herkömmlichen Mobiltelefon dadurch abgegrenzt, dass es zusätzlich zu den reinen Kommunikationsfunktionen, d. h. Telefonate, Short Message Service (SMS) oder Multimedia Messaging Service (MMS), Informationen speichern sowie Apps installieren kann. Auch wenn nach dieser Definition mit dem IBM Simon bereits 1992 das erste Smartphone auf den Markt gebracht wurde, kam der Durchbruch für diese Geräte erst mit der Einführung des ersten iPhones im Jahr 2007 [Sag12]. Neben der wesentlich geringeren Größe, womit das iPhone zum stetigen Begleiter wurde, hat es einen weiteren entscheidenden Vorteil. Durch den Einsatz diverser Sensoren, wie beispielsweise Global Positioning System (GPS), ist es in der Lage, den Kontext des Nutzers zu erfassen und darauf zu reagieren.

Mit jeder Smartphone-Generation stieg bis dato die Anzahl und die Art der verbauten Sensoren an. In Samsungs Galaxy S5 (Modelljahr 2014) finden sich beispielsweise bereits zwei Kameras, drei Mikrophone, ein Gyroskop, ein Beschleunigungssensor, ein Thermometer, ein Fingerabdrucksensor und sogar ein Pulsmessgerät, um nur einige zu nennen [Luc15]. Durch diese Heterogenität der Sensoren sind auch die durch sie ermittelten Daten mannigfaltig. In gleichem Maße verändern sich auch die Einsatzmöglichkeiten; schon lange werden Smartphones nicht mehr nur als Kommunikationsmedium genutzt [FMK+10]. Laut einer Studie von Hampe [Ham15] dienen Smartphones einem Großteil der Anwender zusätzlich u. a. als Foto- bzw. Videokamera (98 % der Befragten), als mobilen Zugang zum Internet (93 % der Befragten) oder als Terminkalender (83 % der Befragten). Weiterhin gaben 74 % der Befragten an, dass sie zusätzliche Apps aus den unterschiedlichsten Anwendungsbereichen (z. B. Fitness-Apps, Apps von sozialen Netzwerken oder Spiele) installieren. Wenig verwunderlich ist es daher, dass die Zahl der Smartphone-Nutzer kontinuierlich steigt [WvdM16] und zudem immer neue Nutzergruppen erschlossen werden [Par14].

Um all diesen verschiedenen Anforderungen der Anwender gerecht werden zu können, wird eine große Zahl an Apps benötigt. Bedingt durch den einfachen Einstieg in die App-Entwicklung und die Möglichkeit Apps z. T. kostenlos über die offiziellen Online-Handelsplattformen, den sogenannten *App-Stores*, veröffentlichen zu können, entwickeln sehr viele Drittanbieter immer neue Apps [Tra12]. In den beiden größten *App-Stores* (*Google Play*[2] und *iTunes*[3]) waren im Jahr 2016 zusammen 4,2 Millionen Apps verfügbar [Stat16]. Ferner wächst die Nachfrage nach Apps kontinuierlich an. Wurden im Jahr 2009 weltweit 2,5 Milliarden Apps aus den offiziellen *App-Stores* heruntergeladen, waren es im Jahr 2016 bereits beinahe 225 Milliarden Apps [Stat13].

Allerdings birgt der Einsatz von Smartphones auch erhebliche Risiken. Auf der einen Seite sorgt die geringe Einstiegshürde für App-Entwickler dafür, dass auch unerfahrene Entwickler Apps veröffentlichen, in denen sich sicherheitsrelevante Schwachstellen befinden. So besitzen viele Apps mehr Berechtigungen als sie eigentlich benötigen, d. h. sie können auf mehr Sensoren oder Daten zugreifen, als es für die Erfüllung ihrer eigentlichen Funktionalität erforderlich ist [FGW11]. Aber selbst erfahrene Software-Entwickler stehen bei der Entwicklung von Apps vor einem Problem: Eine App sollte mindestens für die beiden großen Mobilplattformen iOS und Android verfügbar sein, die im Jahr 2015 zusammen einen Marktanteil von über 95 % ausmachten [LRN15]. Allerdings unterscheidet sich die App-Entwicklung von Plattform zu Plattform sehr stark. Um qualitativ hochwertige Apps entwickeln zu können, müsste der Entwickler die Expertise für alle Zielplattformen besitzen, was nur selten der Fall ist [Tra12]. Auch die Qualitätssicherung hinsichtlich Testfällen ist bei der App-Entwicklung wesentlich eingeschränkt, da Apps häufig nur manuell getestet werden können und Unit-Test-Frameworks für mobile Plattformen nur einen limitierten Funktionsumfang besitzen. Daher ist die Testabdeckung für Apps in der Regel wesentlich geringer, als es bei herkömmlichen Anwendungen für Desktop PCs der Fall ist [JMK13].

Auf der anderen Seite ist aber auch der App-Nutzer eine potentielle Gefahrenquelle. Felt et al. [FHE+12] haben in einer Studie herausgefunden, dass ein Großteil der Nutzer nicht in der Lage ist, die Gefährdung durch die Berech-

[2] siehe https://play.google.com
[3] siehe https://itunes.apple.com/de/genre/ios/id36?mt=8

tigungen, die eine App besitzt, richtig einzuschätzen. So interessieren sich die Nutzer grundsätzlich schon dafür, auf welche Daten eine App zugreifen kann, jedoch sinkt deren Aufmerksamkeit aufgrund des fehlenden Verständnisses rasch. Da sich die Nutzer von Smartphones häufig wesentlich von den Nutzern von Desktop PCs unterscheiden, was sowohl den Einsatzzweck als auch die technische Erfahrung angeht [FMK+10], sind sich diese oftmals der potentiellen Gefahren gar nicht bewusst, die von der unbedachten Verwendung von Apps ausgehen [GVR15]. Aufgrund der Kombination aus privaten Daten, die auf Smartphones gespeichert werden, Apps mit sicherheitskritischen Schwachstellen und unerfahrenen Nutzern geraten Smartphones zunehmend ins Visier von Hackern [Fur05; Sva14].

Hierbei handelt es sich nicht nur um eine theoretische Gefährdung, sondern immer wieder werden private und sensible Daten mittels Apps gestohlen, wie die folgenden Beispiele eindrucksvoll zeigen. Im Jahr 2012 veröffentlicht Bilton [Bil12], dass mit einem Systemupdate für iOS sämtlichen Apps, die Zugriff auf die Standortdaten eines Nutzers haben, der uneingeschränkte Zugriff auf die Fotobibliothek des Nutzers gestattet wurde. Diese Lockerung der Berechtigungen sollte App-Entwicklern die Entwicklung von standortbezogenen Fotoanwendungen erleichtern. Der Nutzer wurde darüber allerdings nicht informiert. Mithilfe einer Test-App wurde nachgewiesen, dass diese App sämtliche Fotos eines Nutzers auf einen Remote-Server kopieren konnte, nachdem ihr vom Nutzer lediglich der Zugriff auf dessen Standortdaten gewährt wurde. Dass es sich hierbei nicht ausschließlich um ein iOS-Problem handelte, zeigten Chen und Bilton [CB12] wenig später. Unter Android benötigte eine App sogar lediglich Zugriff auf das Internet, um die privaten Fotos eines Nutzers zu stehlen. Erschwerend kommt hinzu, dass Google im Jahr 2014 ein Update für die *Google-Play-Store*-App veröffentlicht hat, das die Anzeige der Berechtigungen vollständig neu organisiert. So werden mehrere Berechtigungen in Gruppen zusammengefasst, wodurch die Nutzer nur einen groben Überblick bekommen, welche Berechtigungen einer App erteilt werden. Darüber hinaus werden einige Berechtigungen, die sehr häufig angefordert werden, wie beispielsweise die Berechtigung auf das Internet zugreifen zu können, überhaupt nicht angezeigt, um die Anzeige weiter zu vereinfachen [Nic14]. Aus Nutzersicht würde die

oben genannte App daher augenscheinlich keinerlei Berechtigungen anfordern, aber dennoch auf private Fotos zugreifen können.

Ein weiteres Beispiel, wie über Smartphones Nutzerdaten gestohlen werden können, zeigten die von Edward J. Snowden veröffentlichten Berichte über die Praktiken der National Security Agency (NSA). So gab es mehrere erfolgreiche Versuche seitens der NSA und des Government Communications Headquarters (GCHQ), mittels scheinbar harmlosen Apps an private Informationen über den Nutzer zu gelangen. In einem von Larson, Glanz und Lehren [LGL14] beschriebenen Szenario sollte das Spiel *Angry Birds*[4] dazu genutzt werden, um den Aufenthaltsort, das Alter, das Geschlecht und weitere persönliche Daten des Nutzers in Erfahrung zu bringen. Durch die Injektion von wenigen Zeilen Schadcode ist es möglich, von vielen unterschiedlichen Apps Daten über den Nutzer zu sammeln und so ein detailliertes Profil von ihm zu erstellen. Neben Spielen, in denen viele Nutzer unbedarft ihre Daten angeben, sind Navigationsdienste, wie die *Google-Maps*-App[5], oder Apps von sozialen Netzwerken, wie die *Facebook*-App[6], reichhaltige Informationsquellen [LGL14]. Dass für solche Angriffe keinesfalls die Ressourcen eines Geheimdiensts benötigt werden, zeigt die *Brightest-Flashlight-Free*-App[7]. Diese App benötigt die Berechtigungen, um auf die Standortdaten des Nutzers sowie den Speicher des Smartphones zugreifen zu können, obwohl ihre einzige Funktionalität darin besteht, den Blitz des Smartphones als Taschenlampe zu verwenden. Allerdings muss der Nutzer mit der Verwendung der App deren Endnutzer-Lizenzvereinbarungen zustimmen. In diesen ist festgelegt, dass der Entwickler der App die Daten über den Nutzer sammeln und weiterverkaufen darf [Kas13].

Vordergründig scheinen die obigen Beispiele sehr ähnlich, da jeweils Apps von Drittanbietern genutzt werden, um private Daten über den Nutzer zu sammeln. Betrachtet man die Beispiele allerdings genauer, so lassen sich zwei unterschiedliche Angriffsmuster identifizieren. In den von Bilton [Bil12] und Chen und Bilton [CB12] aufgeführten Angriffen wurden Schwachstellen des Systems bewusst ausgenutzt, um dem Nutzer zu schaden. Die gesammelten

[4] siehe https://play.google.com/store/apps/details?id=com.rovio.angrybirds

[5] siehe https://play.google.com/store/apps/details?id=com.google.android.apps.maps

[6] siehe https://play.google.com/store/apps/details?id=com.facebook.katana

[7] siehe https://play.google.com/store/apps/details?id=goldenshorestechnologies.brightestflashlight.free

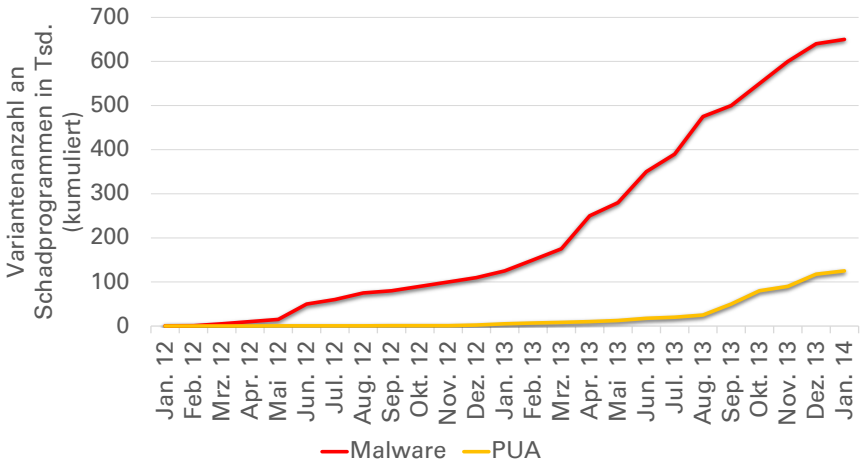


Abbildung 1.1: Kumulatives Wachstum von *Malware* und *PUA* auf mobilen Plattformen (in Anlehnung an [Sva13; Sva14])

Daten wurden illegal und ohne das Wissen oder die Zustimmung des Nutzers erhoben und weiterverarbeitet. In diesen Fällen spricht man von *Malware*, einem aus dem Englischen entnommenen Kunstwort, das sich aus „malicious“ und „software“ zusammensetzt [Sva14]. Larson, Glanz und Lehren [LGL14] sowie Kassner [Kas13] berichten hingegen von Fällen, in denen Apps zu viele Berechtigungen besitzen und diese ausgenutzt werden. Dabei hat der Nutzer diesen Berechtigungen aktiv zugestimmt, vermutlich ohne sich den möglichen Konsequenzen bewusst zu sein. Im Fall der *Brightest-Flashlight-Free*-App stimmte er sogar dem Weiterverkauf der Daten explizit zu. Daher spricht man in diesen Fällen von *Potentially Unwanted Applications (PUAs)*, also Apps, die potentielle Risiken für den Nutzer darstellen, ohne jedoch illegale Aktivitäten durchzuführen [Sva14].

Beide Arten von Schadprogrammen florieren auf mobilen Plattformen, wie in Abbildung 1.1 am Beispiel von Android zu sehen ist. Dabei stellen *PUAs* seit 2014 die am stärksten wachsende Art von Schadprogrammen dar [Qui16].

Während der ungewollte Zugriff auf persönliche Daten wie Fotos, Kontaktdaten oder Standortdaten im privaten Umfeld bereits sehr ärgerlich ist, so können derartige Angriffe noch wesentlich schwerwiegendere Auswirkungen haben,

da Smart Devices immer häufiger im medizinischen Sektor [MHK15] sowie in der Wirtschaft zur Realisierung der *Industrie 4.0* [GGH+13] eingesetzt werden. Auf diese Einsatzszenarien wird in Abschnitt 6.2 respektive Abschnitt 6.3 näher eingegangen.

1.2 Terminologien

Wie die im vorigen Abschnitt beschriebenen erfolgreichen Angriffsszenarien aufzeigen, sind die aktuell zur Verfügung stehenden Sicherheitsmaßnahmen nicht ausreichend, um die Daten der Nutzer angemessen zu schützen. Der Schutz von Daten stellt dabei im Umgang mit Smart Devices eine besondere Herausforderung dar, da durch die Kombination von mehreren Sensordaten neue Informationen über den Nutzer gewonnen werden können. Dies generell zu unterbinden, würde aber eine ungewollte Einschränkung der Funktionsweise dieser Geräte darstellen [Sma15].

Aus rechtlicher Sicht existieren mehrere Gesetze, die das Recht auf informationelle Selbstbestimmung ausreichend festigen [GH16]. In Artikel 8 der Charta der Grundrechte der Europäischen Union ist festgelegt, dass jede Person das Recht auf Schutz der sie betreffenden personenbezogenen Daten besitzt und die Daten nur für die Zwecke verarbeitet werden dürfen, denen die betroffene Person zuvor zugestimmt hat [EU12]. Gemäß Artikel 2 des Übereinkommens zum Schutz des Menschen bei der automatischen Verarbeitung personenbezogener Daten werden als *personenbezogen* sämtliche „Informationen über eine bestimmte oder bestimmbare natürliche Person“ angesehen und unter *Verarbeiten* wird die Speicherung, die Durchführung logischer und / oder rechnerischer Operationen, die Veränderung, das Löschen, die Wiedergewinnung sowie die Bekanntgabe der Daten subsumiert [EU81]. Das Bundesdatenschutzgesetz legt in § 3a darüber hinaus fest, dass bei der Verarbeitung von personenbezogenen Daten auf *Datenvermeidung* und *Datensparsamkeit* zu achten ist [BRD15]. D. h. es darf stets nur ein unbedingt nötiges Mindestmaß an personenbezogene Daten erfasst werden.

Somit stellt der Schutz der Daten in erster Linie ein technisches Problem dar [GH16], zumal eine zu strikte Auslegung der Datensparsamkeit im Widerspruch zu den heutigen Anforderungen steht – der Nutzer profitiert davon,

wenn Apps auf eine Vielzahl an Daten zugreifen können [Hei15]. Erschwerend kommt hinzu, dass der Schutz von Daten häufig mit dem Begriff *Datenschutz* gleichgesetzt wird. Allerdings gilt es zwischen *Datensicherheit*, im Englischen häufig als *Information Security* bezeichnet (siehe Abschnitt 1.2.1), und dem eigentlichen *Datenschutz*, im Englischen häufig als *Data Privacy* bezeichnet (siehe Abschnitt 1.2.2), zu unterscheiden.

Dies lässt sich an den beschriebenen Angriffsszenarien leicht verdeutlichen. Während der Nutzer bei dem Diebstahl seiner privaten Fotos der App Berechtigungen erteilte, diese aber ohne sein Wissen weitreichendere Berechtigungen erlangen konnte, handelt es sich um ein Datensicherheitsproblem, da die Durchsetzung der vereinbarten Berechtigungen nicht eingehalten wurde. Betrachtet man hingegen die *Brightest-Flashlight-Free*-App, so ist dies ein Datenschutzproblem. Hier stimmte der Nutzer der Verwendung seiner Daten zu, ohne die Konsequenzen überblicken zu können. Laut eines Urteils des Bundesverfassungsgerichts vom 15.12.1983 ist es bereits dann eine Gefährdung des Datenschutzes, wenn der Nutzer ausgehend von seiner Wissensbasis nicht in der Lage ist abzusehen, welche Daten durch seine Entscheidungen über ihn bekannt werden [BVG83].

Daher ist eine klare Abgrenzung dieser beiden Begrifflichkeiten nötig, bevor auf technische Maßnahmen zur Realisierung der beiden Schutzaspekte eingegangen werden kann. Im Folgenden werden daher Definitionen für Datensicherheit und Datenschutz gegeben, wie sie im Rahmen dieser Arbeit verwendet werden. Zusätzlich wird auch auf die Wechselwirkung von Datensicherheit und Datenschutz auf das Vertrauen der Nutzer eingegangen und Möglichkeiten aufgezeigt, mit denen das Vertrauen zusätzlich erhöht werden kann. Anzumerken ist, dass im Rahmen dieser Arbeit unter *Sicherheit* die *Informationssicherheit* gemäß der Definition von Freiling et al. [FGG+14] verstanden wird, d. h. sämtliche Maßnahmen zur Reduktion von Gefährdungen für die Daten eines IT-Systems. Die *technische Sicherheit*, d. h. Maßnahmen zum Schutz der Hardware von IT-Systemen, wird nicht betrachtet.

1.2.1 Datensicherheit

Whitman und Mattord [WM12a] definieren vier wesentliche Aufgaben eines Sicherheitssystems. Diese sind die Aufrechterhaltung der Funktionalität eines

Systems, die sichere Ausführung von Operationen auf dem System, der Schutz vor Datendiebstahl und die Fähigkeit, sich auf neue Situationen und Aufgaben einstellen zu können. Im Bereich der Datensicherheit bedeutet dies, dass die Daten dauerhaft zur Verfügung stehen, Apps keinen illegalen Zugriff auf diese Daten erlangen oder die Daten heimlich an Dritte weitergeben können und dass bei Bedarf neue Datenquellen unterstützt werden. Das Ziel all dieser Sicherheitsmaßnahmen ist der Schutz vor illegalen Aktivitäten. Der Angreifer kann dabei von innen (z. B. durch *Malware*) oder von außen (z. B. bei Systemen mit mehreren Nutzern) agieren. Technisch können diese Funktionen beispielsweise durch Kryptographie, Zugriffskontrollsysteme oder Nutzerautorisierung realisiert werden. Datensicherheitssysteme benötigen keinerlei Interaktion mit dem Nutzer, da sie dauerhaft autonom im Hintergrund arbeiten und das System schützen. Obwohl Datensicherheit eigentlich oberste Priorität für jeden Nutzer besitzen sollte, wird dieses Thema besonders auf mobilen Plattformen häufig komplett vernachlässigt [GZZ+12].

1.2.2 Datenschutz

Agre und Rotenberg [AR98] beschreiben *Privatheit* (englisch *Privacy*) als die Möglichkeit in jeder Art von sozialen Beziehungen den Zugriff auf persönliche Daten zu regulieren. Übertragen auf einen technischen Kontext beschreibt Datenschutz demnach übereinstimmend mit den gesetzlichen Reglementierungen Maßnahmen, mit denen der Zugriff von Apps auf private Informationen kontrolliert werden kann. Unter einem Zugriff wird die Speicherung, die Verarbeitung, die Veränderung, das Löschen, die Wiedergewinnung sowie die Bekanntgabe der Daten verstanden. Technisch werden diese Funktionen beispielsweise von einem Berechtigungssystem realisiert, das es dem Anwender ermöglicht Regeln zu definieren, die den Zugriff auf die privaten Daten regulieren. Ziel des Datenschutzes ist es zu definieren, welche Aktivitäten zulässig sind und welche als illegal zu klassifizieren sind [SDX11]. Die Durchsetzung dieser Regeln fällt nicht unter den Aspekt Datenschutz sondern ist Aufgabe eines Datensicherheitssystems gemäß der Definition aus Abschnitt 1.2.1.

Die Regeln können nur vom Nutzer selbst definiert werden, da das richtige Maß an Privatheit im Auge des Betrachters liegt. Er muss festlegen, wie viele und welche Daten er bereit ist mit einer App zu teilen sowie auf welche Art und

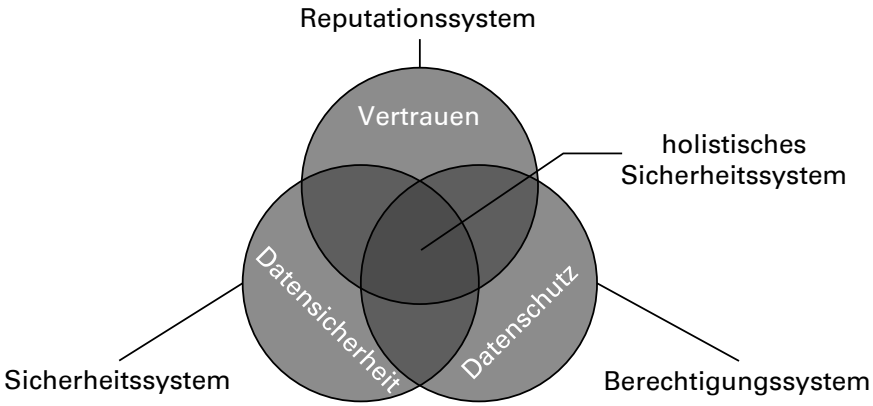


Abbildung 1.2: Zusammenhang zwischen Datensicherheit, Datenschutz und Vertrauen (in Anlehnung an [SdLL10])

zu welchem Zweck diese App die Daten verarbeiten darf [Ved11]. Allerdings kann er bei seiner Entscheidung durch das Berechtigungssystem unterstützt werden, z. B. indem ihm Handlungsempfehlungen aufgezeigt werden. Dadurch ist es möglich, den potentiellen Schaden, der von PUAs ausgehen kann, zu reduzieren.

Offensichtlich hängen trotz all dieser Unterschiede Datensicherheit und Datenschutz eng zusammen. Ein holistisches Sicherheitssystem muss daher beide Aspekte berücksichtigen. Der Nutzer benötigt ein Berechtigungssystem, mit dem er Datenschutzregeln festlegen kann, und ein Sicherheitssystem, das seine Daten vor illegalen Angriffen schützt. Abbildung 1.2 bildet diesen Zusammenhang ab.

Zusätzlich ist es hilfreich, wenn der Nutzer einschätzen kann, inwiefern er einer App vertrauen kann. Zwischen Vertrauen und Datensicherheit sowie Datenschutz besteht ebenfalls eine Wechselwirkung: Wenn ein Sicherheitssystem zuverlässig arbeitet, dann kann der Nutzer auch Apps, denen er nicht vertraut, bedenkenlos ausführen, da selbst bösartige Apps keinen Schaden anrichten können. So sieht Heidkamp [Hei15] eine sichere IT (im Sinne von Datensicherheit und Datenschutz) als Grundlage für ein steigendes Vertrauen der Nutzer in die Technik. Wenn garantiert wird, dass Apps nur über klar definierte

Schnittstellen auf Daten zugreifen können und der Zugriff über diese Schnittstellen durch den Nutzer reglementiert werden kann, so steigt automatisch das Vertrauen in das Gesamtsystem. Da es allerdings keinen vollständigen Schutz geben kann, lohnt es sich weiterführende, vertrauensbildende Maßnahmen zu berücksichtigen.

1.2.3 Vertrauen

Ran et al. [RSC+11] definieren Vertrauen als eine auf Erfahrungen beruhende Messgröße, die angibt, wie zuverlässig man Entscheidungen treffen kann. Diese Entscheidungen können aufgrund von drei wesentlichen Einflussfaktoren getroffen werden: Die **subjektiven und objektiven Eigenschaften des Nutzers** (z. B. dessen Zutrauen respektive ihm zugesicherte Standards), die **äußeren Umstände** (z. B. eine Nutzen-Risiko-Abwägung) und die **subjektiven und objektiven Charakteristika der App oder des App-Entwicklers** (z. B. dessen Motivation respektive dessen Reputation) [YH13]. Vertrauen ist ein komplexes Maß, da es hochgradig subjektiv, kontext-abhängig, asymmetrisch und nur teilweise transitiv ist [AHH+10; KK11]. Obwohl Vertrauen subjektiv ist, ist es einem Nutzer in einem komplexen Umfeld, wie es die heutige IT-Landschaft darstellt, kaum möglich, sich selbstständig eine klare Meinung zu bilden. Daher wird verstärkt auf ein *verteiltes Vertrauensmodell* gesetzt, in dem jeder Teilnehmer seine Handlungsempfehlungen mit anderen Teilnehmern teilt und diese die Empfehlungen zu einem gewissen Maß bei ihren Entscheidungen berücksichtigen können [KFJ01].

Sicari et al. [SRGC15] stellen in ihrer Vergleichsstudie fest, dass im Umfeld von Smart Devices die Geräte einen starken Bezug zu einem festen Nutzer haben. Daher muss bei einer Vertrauensmetrik neben dem Vertrauen in Software- oder Hardwarekomponenten auch soziale Beziehungen zwischen den Nutzern abgebildet werden können. So sollte eine Datenanfrage unterschiedlich bewertet werden, abhängig davon, ob sie von einem befreundeten oder einem unbekannten Nutzer kommt. Dies erhöht die Komplexität der Vertrauensbewertung noch weiter. Eine solche Metrik kann technisch als Reputationssystem realisiert werden, wie es von Ben Saied et al. [BOZL13] für verteilte Systeme vorgeschlagen wird. Dabei wird dem Nutzer aufgrund mehrerer Faktoren auf-

gezeigt, wie vertrauenswürdig eine anfragende Instanz im gegebenen Kontext ist.

Die vorliegende Arbeit fokussiert sich allerdings auf technische Lösungen zur Erhöhung der Datensicherheit und des Datenschutzes, da auf diese Weise bereits das Vertrauen gestärkt wird. Dennoch wurden in der Fachstudie von Aukschlat, Jasny und Pirk [AJP14] sowie in der Bachelorarbeit von Aukschlat [Auk14] bereits Möglichkeiten untersucht, wie eine Vertrauensmetrik in die in dieser Arbeit vorgestellten Datensicherheits- und Datenschutzlösungen integriert werden kann. Dadurch kann beispielsweise dem Nutzer geholfen werden, die für seine Einsatzzwecke bestmöglichen Berechtigungseinstellungen vorzunehmen.

1.3 Schutzziele

In dem ISO-Standard ISO/IEC 27001:2013 [ISO13b] wird festgelegt, wie ein Datensicherheitssystem aufgebaut sein muss und welche Anforderungen erfüllt sein müssen, um Datensicherheit garantieren zu können. Dabei rückt der Standard in der Überarbeitung aus dem Jahr 2013 die Information als wichtigstes schützenswertes Gut in den Fokus [Jen14]. Klipper [Kli15] identifiziert hierbei die drei Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit als Grundwerte. Die nachfolgenden Definitionen der Schutzziele für Datensicherheit folgen dem ISO-Standard ISO/IEC 27000:2009 [ISO09].

[IS1] – Vertraulichkeit:

Es muss sichergestellt werden, dass die Daten keinen unautorisierten Instanzen (sowohl Personen als auch Prozessen) zugänglich gemacht werden.

[IS2] – Integrität:

Es muss sichergestellt werden, dass die Daten von Dritten nicht verändert werden (weder hinsichtlich der Genauigkeit noch der Vollständigkeit).

[IS3] – Verfügbarkeit:

Es muss sichergestellt werden, dass die Daten autorisierten Instanzen dauerhaft zur Verfügung stehen.

Pohl [Poh04] sieht in der Genauigkeit eine nötige Verfeinerung und Ergänzung des Schutzziels Verfügbarkeit.

[IS4] – Genauigkeit:

Es muss sichergestellt werden, dass die Daten in der benötigten Genauigkeit vorliegen.

Eckert [Eck13] ergänzt diese grundlegenden Schutzziele um die folgenden drei Werte:

[IS5] – Nachvollziehbarkeit:

Es muss sichergestellt werden, dass sämtliche Aktivitäten, wie beispielsweise Zugriffe auf Daten, persistent und unüberwindbar überwacht werden.

[IS6] – Authentizität:

Es muss sichergestellt werden, dass die Instanzen, die Daten anfordern, eindeutig identifiziert werden.

[IS7] – Verbindlichkeit:

Es muss sichergestellt werden, dass eindeutig belegbar ist, ob eine Instanz bestimmte Daten verarbeitet hat.

Um die Instanzen eindeutig identifizieren zu können und somit die Nachvollziehbarkeit, Authentizität und Verbindlichkeit garantieren zu können, wird ein weiteres Schutzziel, die Autorisierung, benötigt [BSI12].

[IS8] – Autorisierung:

Es muss sichergestellt werden, dass ein Mechanismus zur Verfügung steht, mit dem sich Instanzen gegenüber dem Datensicherheitssystem autorisieren können.

Besonders im Umgang mit privaten und sensiblen Daten, ist es darüber hinaus wichtig, dass der Nutzer die Möglichkeit besitzt, bestimmte Daten zu verfälschen oder die Genauigkeit zu reduzieren, um diese so vor einer ungewollten Verwendung zu schützen [WP00].

[IS9] – Pseudonymität:

Es muss sichergestellt werden, dass ein Nutzer einen Service nutzen kann, ohne seine vollständige Identität preiszugeben, z. B. indem er ausgewählte private Daten verfremdet an den Service sendet. Die Schutzziele [IS1] bis [IS8] dürfen hiervon allerdings nicht kompromittiert werden.

In einem holistischen Sicherheitssystem reicht die alleinige Realisierung dieser Schutzziele der Datensicherheit nicht aus. Es ist ebenfalls wichtig, dass das System technische Aspekte berücksichtigt, die den **Datenschutz** gewährleisten, d. h. die die Datenschutzgesetze praktisch umsetzen [CH13].

In Anlehnung an das *Montreux-Dokument* [DSK05], Wagner und Raabe [WR16] und Schallaböck [Sch16] werden im Rahmen dieser Arbeit folgende sechs Schutzziele $DP1$ bis $DP6$ erhoben, die von einem Datenschutzsystem zu erbringen sind. Jedem Schutzziel liegen mehrere europäische Vorgaben oder nationale Gesetze zugrunde, wobei jeweils ein Beispiel aufgeführt wird.

[DP1] – Einwilligung:

Für die Verarbeitung der Daten muss eine Einwilligung der betroffenen Person vorliegen (u. a. gemäß § 4 Absatz 2 des Bundesdatenschutzgesetzes [BRD15]).

[DP2] – Zweckbindung:

Die erhobenen personenbezogenen Daten dürfen nur für den vereinbarten Zweck verwendet und nicht unkontrolliert weiterverwendet werden (u. a. gemäß Artikel 6 Absatz 1b der Europäischen Datenschutzrichtlinie [EU95]).

[DP3] – Erforderlichkeit:

Die erhobenen personenbezogenen Daten sind zur Erfüllung der Funktionalität notwendig (u. a. gemäß Artikel 8 der Charta der Grundrechte der Europäischen Union [EU12]).

[DP4] – Transparenz:

Der Nutzer muss ausreichend über die Verwendung seiner Daten informiert werden, damit er in der Lage ist, entscheiden zu können, ob er dieser Verwendung zustimmen will (u. a. gemäß Artikel 12 der Datenschutz-Grundverordnung [EU16]).

[DP5] – Datensparsamkeit:

Die Menge der erhobenen Daten wird auf ein Mindestmaß reduziert (u. a. gemäß § 3a des Bundesdatenschutzgesetzes [BRD15]).

[DP6] – Kontrolle:

Es werden Mechanismen benötigt, damit der Nutzer die Kontrolle über seine persönliche Daten behält (u. a. gemäß § 1 Absatz 1 und 2 des Bundesdatenschutzgesetzes [BRD15]).

Darüber hinaus muss auch **Datensicherheit** gewährleistet sein, da durch Lücken in der IT-Sicherheit sämtliche Schutzziele bezüglich des Datenschutzes ausgehebelt werden könnten [WR16]. Setzt ein Datenschutzsystem diese Schutzziele und damit die rechtlichen Vorgaben um, so wird dadurch die **Rechtmäßigkeit** der Datenverarbeitung sichergestellt [Sch16].

1.4 Forschungsbeiträge

Da die *Informationsrevolution* unsere Lebensführung nachhaltig verändert und die Vielfalt und Dynamik der IT-Landschaft stetig zunimmt, erhöht sich die Komplexität, mit der Datensicherheits- und Datenschutzsysteme umgehen müssen. Gleichzeitig wird für unsere *digitale Gesellschaft* der Schutz von Daten immer essenzieller. Hierfür werden flexible Sicherheitssysteme benötigt, die vielseitig und erweiterbar sind, um mit der Dynamik der IT-Systeme mithalten zu können [Hol14]. Weder der Stand der Technik noch der Stand der Forschung bietet ein solches System, das alle in Abschnitt 1.3 identifizierten Schutzziele erfüllt [SM13; SM15; SM16a].

Der wesentliche Beitrag der vorliegenden Arbeit ist daher die Konzeption, prototypische Implementierung und Evaluation eines holistischen Sicherheits-

systems für Smart Devices. Zu diesem Zweck werden die nachfolgenden fünf Forschungsbeiträge erbracht. Für jeden Forschungsbeitrag sind die eigenen Hauptpublikationen angegeben; auf die weiteren eigenen Publikationen zu den einzelnen Forschungsbeiträgen wird in den zugehörigen Kapiteln verwiesen.

Analyse bestehender Datensicherheits- und Datenschutzkonzepte (FB_1):

Das Ziel ist es die aktuell vorherrschenden mobilen Plattformen hinsichtlich ihrer Datensicherheits- und Datenschutzkomponenten zu analysieren (siehe Kapitel 2). Dabei beschränkt sich diese Arbeit auf die drei Marktführer Apple iOS, Microsoft Windows Phone und Google Android, da sie zusammen im Jahr 2015 einen Marktanteil von 99,3 % unter den mobilen Plattformen ausmachten [LRN15].

Diese Analyseergebnisse bilden die Grundlage der vorliegenden Arbeit. Darauf aufbauend können verbesserte Datensicherheits- und Datenschutzkonzepte erstellt, die die in Abschnitt 1.3 ermittelten Schutzziele erfüllen, sowie Implementierungsstrategien für diese erarbeitet werden (siehe Kapitel 3).

Die Hauptpublikation für diese Analyse stellt [Sta13b] dar.

Erstellung eines kontextsensitiven Berechtigungsmodells (FB_2):

Das Ziel ist die Erarbeitung eines alternativen Berechtigungsmodells für mobile Plattformen namens *Privacy Policy Model (PPM)* (siehe Abschnitt 4.1). Dieses Berechtigungsmodell muss erweiterbar sein, um ebenfalls Sensoren in zukünftigen Smartphone-Generationen unterstützen zu können, und es müssen Kontextinformationen bei den Berechtigungsregeln berücksichtigt werden.

Das Berechtigungsmodell bildet die Grundlage für das im Kern dieser Arbeit stehende Datenschutzsystem.

Die Hauptpublikation für dieses Berechtigungsmodell stellt [SM13] dar.

Einbettung und Implementierung des Berechtigungsmodells in einem flexiblen Datenschutzsystem (FB_3):

Das Ziel ist die Realisierung des Berechtigungsmodells in einem Daten-

schutzsystem namens *PMP* (siehe Kapitel 4). Dieses Datenschutzsystem muss die in Abschnitt 1.3 aufgestellten Schutzziele [DP1] bis [DP6] berücksichtigen und auf die besonderen Herausforderungen im Bereich mobiler Plattformen eingehen. Insbesondere bedeutet dies, dass das System feingranulare, kontextsensitive Berechtigungsregeln unterstützt, die sich zur Laufzeit vom Nutzer anpassen lassen und erweiterbar sind.

Die Hauptpublikation für dieses Datenschutzsystem stellen [SM14] und [Sta15a] dar.

Erweiterung des Datenschutzsystems zu einem holistischen Sicherheitssystem (FB_4):

Das Ziel ist die Realisierung eines holistischen Sicherheitssystems, welches neben Datenschutz- auch Datensicherheitsfeatures anbietet. Hierzu soll die *PMP* um eine Datensicherheitskomponente namens *SDC* erweitert werden (siehe Kapitel 5). Der *SDC* muss die in Abschnitt 1.3 aufgestellten Schutzziele [IS1] bis [IS9] berücksichtigen.

Die Hauptpublikation für diese Datensicherheitskomponente stellt [SM15] dar.

Evaluation des holistischen Sicherheitssystems (FB_5):

Das Ziel ist die Evaluation der *PMP* und deren Erweiterung *SDC* (siehe Kapitel 7). Dabei wird beurteilt, inwiefern dieses holistische Sicherheitssystem die Schutzziele erfüllt, wie hoch der Ressourcenbedarf des Systems ist und wie gut die Handhabbarkeit für Nutzer und Entwickler ist. Ferner wird das Sicherheitssystem auch mit dem Stand der Technik sowie dem Stand der Forschung kritisch verglichen.

Die Hauptpublikation für diese Evaluation stellt [SM16a] dar.

Neben diesen fünf Forschungsbeiträgen präsentiert die vorliegende Arbeit auch Anwendungsbeispiele für die *PMP* und den *SDC* aus der Praxis (siehe Kapitel 6) und stellt die im Rahmen der Arbeit konzipierte und realisierte Werkzeugunterstützung für die *PMP* vor (siehe Kapitel 8).

Abbildung 1.3 zeigt, wie Forschungsbeiträge (FB_2), (FB_3) und (FB_4) konzeptionell realisiert werden sollen. Die *PMP* agiert als logische Zwischenschicht

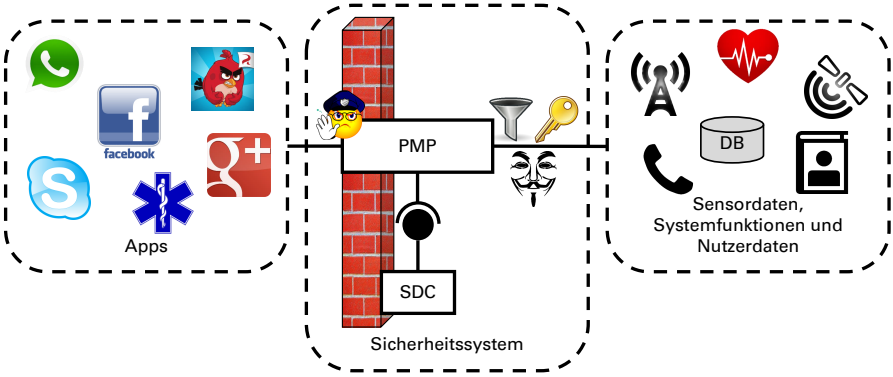


Abbildung 1.3: Konzeptionelles Modell der *PMP* und des *SDCs*

zwischen den Apps auf der einen und Sensordaten, Systemfunktionen und Nutzerdaten auf der anderen Seite. Wenn eine App auf Daten oder Funktionen zugreifen will, so muss sie diese bei der *PMP* anfragen. Diese prüft in den vom Nutzer aufgestellten Berechtigungsrichtlinien gemäß des *PPMs*, ob der Zugriff zulässig ist. Zusätzlich kann die *PMP* bei Bedarf sämtliche Daten filtern, verfremden oder verschlüsseln, bevor diese an eine App weitergegeben werden. Der *SDC* ist als Plugin für den Nutzer transparent an die *PMP* angeschlossen und bietet zusätzlich Datensicherheitsfeatures.

Obwohl die in der vorliegenden Arbeit vorgestellten Prototypen der *PMP* sowie des *SDCs* für Android umgesetzt wurden, lassen sich die zugrunde liegenden Sicherheitskonzepte leicht auf andere IT-Umgebungen übertragen, die mit einer Vielzahl an sensiblen Daten umgehen. Beispiele hierfür sind die App-Plattform *Facebook Platform*[8], das Betriebssystem *Chromium OS*[9] oder die *Internet of Things (IoT)*-Plattform *Amazon Web Services (AWS)*[10].

1.5 Aufbau der Arbeit

Die vorliegende Arbeit ist folgendermaßen aufgebaut:

[8] siehe http://www.facebook.com/platform
[9] siehe https://www.chromium.org/chromium-os
[10] siehe https://aws.amazon.com

In **Kapitel 2 – Analyse bestehender mobiler Plattformen** – werden mit Apples iOS, Microsofts Windows Phone und Googles Android die drei aktuell vorherrschenden Mobilplattformen näher beleuchtet. Dabei werden in erster Linie Eigenschaften berücksichtigt, die einen direkten Einfluss auf die Datensicherheit und den Datenschutz (im Verständnis dieser Arbeit, siehe Abschnitt 1.2) haben.

In **Kapitel 3 – Analyse von Android-Datenschutzsystemen** – werden Möglichkeiten betrachtet, wie Android um Sicherheitsaspekte erweitert werden kann, um die Datensicherheit und den Datenschutz zu verbessern. Android dient hierbei lediglich als ein Beispiel für eine Mobilplattform. Die erarbeiteten Konzepte können ebenfalls auf jede andere IT-Umgebung übertragen werden, die mit einer Vielzahl an sensiblen Daten umgeht.

In **Kapitel 4 – Die Privacy Management Platform** – wird mit der *PMP* ein holistisches Datenschutzsystem vorgestellt. Zu diesem Zweck wird zunächst auf das *PPM* eingegangen, mit dem der Datenzugriff von Apps beschränkt werden kann. Das *PPM* bildet das Kernstück der *PMP*. Anschließend werden auf relevante Charakteristika der *PMP* eingegangen. Basierend auf diesen Grundlagen, werden unterschiedliche Implementierungsstrategien für die *PMP* diskutiert und mit dem *PMP Gatekeeper* eine *Best-of-Breed*-Implementierungsstrategie herausgearbeitet.

In **Kapitel 5 – Der Secure Data Container** – wird der *SDC*, ein generischer Datenspeicher, vorgestellt. Der *SDC* ist eine Erweiterung für die *PMP* und ermöglicht es Apps, Daten sicher und effizient untereinander auszutauschen. Da Android für Smartphones auf einen reinen Einbenutzerbetrieb ausgelegt ist, führt der *SDC* Nutzerkonten ein, um auch einen sicheren Datenaustausch zwischen unterschiedlichen Nutzern zu ermöglichen. Zusätzlich wird mit dem *SDC* die Datensicherheit weiter erhöht, indem sämtliche Daten darin nur verschlüsselt vorliegen.

In **Kapitel 6 – Anwendungsbeispiele für die PMP** – werden fünf Anwendungsbeispiele aus unterschiedlichen Domänen (u. a. *mHealth*, *IoT* oder der *Industrie 4.0*) vorgestellt. Anhand dieser Anwendungsbeispiele werden die Mächtigkeit sowie die Verwendungsmöglichkeiten der *PMP* demonstriert.

In **Kapitel 7 – Evaluation der PMP und des SDCs** – wird sowohl die *PMP* als auch der *SDC* unter mannigfaltigen Aspekten evaluiert. Einerseits wird

überprüft, inwiefern die *PMP* respektive die Kombination aus *PMP* und *SDC* die in Abschnitt 1.3 aufgestellten Schutzziele erfüllt. Andererseits werden diese beiden Systeme unter technischen Gesichtspunkten analysiert und deren Ressourcenverbrauch bestimmt. Auch eine Bewertung aus Anwendersicht (sowohl aus Nutzer- als auch aus App-Entwicklersicht) wird durchgeführt. Schließlich wird die *PMP* und der *SDC* mit dem Stand der Technik sowie dem Stand der Forschung verglichen.

In **Kapitel 8 – Werkzeugunterstützung für die PMP** – werden Werkzeuge vorgestellt, welche die Entwicklung von Apps für die *PMP* erleichtern. Hierbei handelt es sich um Plugins für die Entwicklungsumgebung *eclipse*, mit denen Metadaten für Apps erstellt werden können, die für die *PMP* benötigt werden. Außerdem wird ein Entwurf eines Frameworks für auto-adaptive Apps vorgestellt, damit diese selbstständig auf einen eingeschränkten Datenzugriff reagieren können (z. B. indem der Funktionsumfang entsprechend eingeschränkt wird).

In **Kapitel 9 – Zusammenfassung und Ausblick** – wird ein Fazit gezogen und ein Ausblick darauf gegeben, wie die vorgestellten Forschungsbeiträge auch auf weitere Forschungsbereiche angewandt werden können. Insbesondere der Transfer der Datenschutzkonzepte auf *Complex Event Processing (CEP)* Systeme (siehe Mühl, Fiege und Pietzuch [MFP06]) steht hierbei im Fokus.

Kapitel 2

Analyse bestehender mobiler Plattformen

Aktuell dominieren Google mit Android (84,1 % Marktanteil im ersten Quartal 2016) und Apple mit iOS (14,8 % Marktanteil im ersten Quartal 2016) den Markt der Mobilplattformen [WvdM16]. Um Schwächen bezüglich Datensicherheit und Datenschutz, die in mobilen Plattformen existieren, identifizieren zu können, ist es unumgänglich, sich eingehend mit diesen beiden Plattformen zu beschäftigen. Auch wenn Microsoft mit Windows Phone kaum eine Rolle auf dem Markt spielt (0,7 % Marktanteil im ersten Quartal 2016) [WvdM16], sollte diese Plattform in einer Analyse dennoch berücksichtigt werden. Im Gegensatz zu den beiden anderen Plattformen basiert Windows Phone nicht auf einem Linux Kernel, sondern auf einem Windows Kernel und bringt somit von Grund auf andere Voraussetzungen mit [GHG10]. Insgesamt sind diese drei Plattformen hochgradig heterogen und setzen stark divergente Sicherheitsmodelle um [GHGY14].

Zunächst werden daher die sicherheitsrelevanten Funktionen von iOS[1] (Abschnitt 2.1), von Windows Phone[2] (Abschnitt 2.2) und von Android[3] (Abschnitt 2.3) vorgestellt. Die Analyse beschränkt sich dabei nicht auf eine spezielle Version der Plattformen. Bei versionsspezifischen Funktionen oder Eigenschaften wird dies an der jeweiligen Stelle explizit vermerkt. Weiter ist anzumerken, dass die Analyse sich mit Windows Phone und nicht mit dessen Vorgänger Windows Mobile beschäftigt. Trotz Namensähnlichkeit handelt es sich hierbei um zwei vollständig eigenständige Produkte, wobei Windows Mobile seinem Nachfolger unterlegen ist [BK11] und seitens Microsoft mit der Einführung von Windows Phone nicht länger weiterentwickelt wird [Mic16]. Anschließend werden die drei Plattformen miteinander verglichen (Abschnitt 2.4) und deren Sicherheitsmodelle klassifiziert (Abschnitt 2.5). Dies bildet die Wissensbasis dafür, inwiefern Anpassungen an den Sicherheitssystemen mobiler Plattformen nötig sind, um die in Abschnitt 1.3 aufgestellten Schutzziele zu unterstützen. Darüber hinaus liefert es eine Entscheidungshilfe, für welche der Plattformen die in der vorliegenden Arbeit vorgestellten Konzepte prototypisch umgesetzt werden sollten.

Dieses Kapitel basiert auf den eigenen Publikationen [Sta13b] sowie [SM13], wobei die Inhalte stark erweitert und auf die zwischenzeitlichen Veränderungen an den Plattformen angepasst wurden.

2.1 iOS von Apple

Hinsichtlich Datensicherheit bietet iOS den Nutzern eine 360-Grad-Lösung. Angefangen beim Systemstart über die Installation von Apps bis zu deren Ausführung greifen die einzelnen Konzepte nahtlos ineinander, um dadurch Angreifern eine möglichst geringe Angriffsfläche zu bieten [Eil14]. Im Folgenden werden die wichtigsten Bestandteile dieser Schutzkette vorgestellt.

Abbildung 2.1 bildet die Sicherheitsarchitektur ab, die in der iOS-Plattform zum Einsatz kommt. Beim Systemstart wird zunächst der Startcode geladen. Dieser beinhaltet u. a. das Apple-Root-Zertifikat, mit dem die Echtheit von allen Softwarekomponenten, wie beispielsweise Apps, verifiziert werden kann.

[1] siehe http://www.apple.com/de/ios/
[2] siehe https://www.microsoft.com/de-de/windows/phones
[3] siehe https://www.android.com/

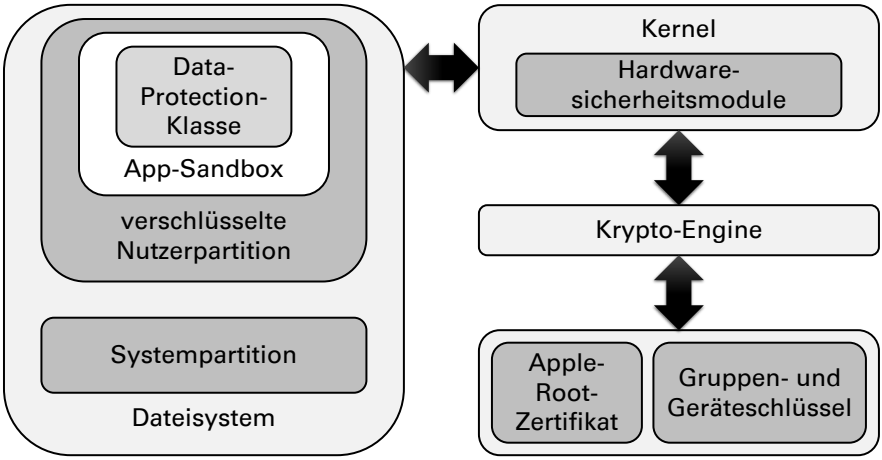


Abbildung 2.1: Die iOS-Sicherheitsarchitektur (in Anlehnung an [App16; Eil14])

Der Startcode liegt in einem speziellen Read-Only Memory (ROM)-Chip und ist damit unveränderlich. Mithilfe dieses Zertifikats kann anschließend der eigentliche Boot-Prozess schrittweise gestartet und in jedem Schritt die geladenen Komponenten überprüft werden. Wird bei einer Komponente eine Manipulation festgestellt, so kann das System nicht gestartet werden und das System muss auf den Werkszustand zurückgesetzt werden. Dies beinhaltet eine vollständige Neuinstallation der Systemkomponenten, die über den offiziellen *App-Store* von Apple bezogen werden müssen. Modifikationen an Systemkomponenten durch Dritte, insbesondere aber auch durch den Nutzer, sind nicht zulässig. Mittels dieses Verifikationsprozesses stellt Apple darüber hinaus sicher, dass auf iOS-Geräten immer die aktuellste iOS-Version installiert ist, wodurch Sicherheitslücken zeitnah geschlossen werden können [App16].

Nicht nur die Systemkomponenten, sondern auch jede App (unabhängig davon ob es sich um eine offizielle App von Apple oder eine App eines Drittanbieters handelt) wird vom System verifiziert. Eine App muss einerseits von dem Entwickler und andererseits von Apple digital signiert sein, um ausführbar zu sein. Durch die Signatur des Entwicklers kann der Ursprung einer App überprüft werden und es kann sichergestellt werden, dass die App nicht nach-

träglich manipuliert wurde. Bevor eine App die erforderliche Signatur von Apple erhält, wird die App hinsichtlich mehrerer sicherheitsrelevanter Gesichtspunkte analysiert. Die Signatur von Apple dient demnach als Gütesiegel, dass es sich bei der App um keine Schadsoftware handelt [Mil11].

Unter iOS läuft jede App in einer eigenen Sandbox (siehe Abbildung 2.1). Die Idee ist, dass innerhalb der Sandbox nur die Daten und Funktionen zur Verfügung stehen, die für die Ausführung der App benötigt werden. Insbesondere muss sichergestellt werden, dass jede Sandbox exakt die gleichen Berechtigungen wie die App selbst besitzt [PS01]. Jede App wird zu diesem Zweck bereits bei der Installation zusammen mit einer Konfigurationsdatei in ein separates `home`-Verzeichnis kopiert. Nur auf dieses `home`-Verzeichnis und dessen Unterverzeichnisse hat die App Zugriffsrechte; auch der potentielle Schaden, der von einer App ausgeht ist demzufolge auf die Daten in diesen Verzeichnissen beschränkt. Die Sandbox isoliert dadurch die App gleichermaßen von anderen Apps wie auch von dem System an sich. Hardwaresicherheitsmodule stellen diese Isolation sicher. So sorgt *Address Space Layout Randomization (ASLR)* dafür, dass die Speicherbereiche zufällig an Apps verteilt werden, und *Execute Never (XN)* garantiert, dass Apps, die von Apple nicht in die höchste Schutzklasse einsortiert wurden, nicht gleichzeitig Schreib- und Ausführrechte auf einen Speicherbereich bekommen [Eil14].

Damit eine App dennoch auf Systemdaten (z. B. die Kontakte oder Sensordaten) zugreifen kann, stellt iOS verschiedene Application Programming Interfaces (APIs) zur Verfügung. Der Funktionsumfang dieser APIs ist unter der mobilen iOS-Version auf ein Minimum reduziert [Sad09]. Darüber hinaus lassen sich über die APIs keine Manipulationen an anderen Apps durchführen. Ein direkter Austausch von Daten zwischen mehreren Apps ist unter iOS nicht möglich, aber es stehen Systemfunktionen zur Verfügung, über die mehrere Apps auf den gleichen Speicherbereich zugreifen können (z. B. über gemeinsame Schlüsselbundbereiche [Dai11]) [DSK11].

Apple liefert alle Geräte von Haus aus mit einer Hardwarekomponente zur effizienten Ver- und Entschlüsselung von Daten aus. Von dieser Krypto-Engine werden unterschiedliche Verschlüsselungsalgorithmen wie Advanced Encryption Standard (AES) unterstützt. Als Schlüssel dient der eindeutige Geräteschlüssel, der unveränderbar in den Prozessor eingebrannt ist. Dadurch

wird sichergestellt, dass die Daten von keinem anderen Gerät gelesen werden können [Eil14].

Zusätzlich kann für jede Datei individuell festgelegt werden, welcher Data-Protection-Klasse diese zugeordnet ist. Diese Klasse bestimmt, wann eine Datei verschlüsselt wird bzw. wie lange sie unverschlüsselt bleiben darf [Mor10]. Durch eine konsequente Aufspaltung der Daten in eine Systempartition und eine Nutzerpartition können für die Systemdaten weitere Schutzmaßnahmen getroffen werden, z. B. dass die Systempartition nur mit Leserechten eingebunden wird [App16].

Während all diese Sicherheitsmaßnahmen für den Nutzer transparent sind, kann er nur über den Sperrcode direkt Einfluss auf die Datensicherheit nehmen. Wird das Gerät gesperrt, so werden alle Daten gemäß der zugeordneten Data-Protection-Klasse gesperrt. Nur mit der korrekten Personal Identification Number (PIN) kann das Gerät wieder entsperrt und die Daten dadurch wieder lesbar gemacht werden. Diese Maßnahme richtet sich ausschließlich gegen physische Angriffe auf das Gerät. Darüber hinaus kann der Nutzer aus der Ferne die Schlüssel löschen und somit seine Nutzerdatenpartition dauerhaft unlesbar zu machen [HS11].

Dieser umfassende 360-Grad-Schutz kann nur gewährleistet werden, da Hard- und Softwarekomponenten nahtlos ineinander greifen. Dies kann nur sichergestellt werden, da Apple beides herstellt und iOS nur auf offiziellen Apple-Geräten lauffähig ist. Obwohl die Sicherheitsmaßnahmen umfassend sind, kann jede der Komponenten erfolgreich angegriffen werden [WLL+13]. Die Zahl an Schadsoftware die in *iTunes* angeboten wird, steigt trotz aller Vorkehrungen stetig an [UC16]. Auch haben Untersuchungen gezeigt, dass Apps aus ihrer Sandbox ausbrechen können und somit illegal Zugriff auf andere Apps sowie deren Daten erhalten [XBL+15]. Insgesamt betrachtet, ist die Anzahl an existierender Schadsoftware im Vergleich zu anderen mobilen Plattformen aber dennoch relativ gering [Kel14].

2.2 Windows Phone von Microsoft

Microsoft stellte bei den ersten mobilen Windows-Versionen noch sehr viele Restriktionen an die Hardwarehersteller. Dadurch war die Vielfalt an unterstütz-

ten Geräten stark eingeschränkt (wenn auch nicht in dem Ausmaß, wie es bei Apple der Fall ist). Diese Haltung ändert sich langsam und Microsoft öffnet sich mehr und mehr gegenüber Hardwareherstellern und -plattformen [Whi15]. Im Gegensatz zu den beiden anderen untersuchten mobilen Plattformen basiert Windows Phone nicht auf einem Linux Kernel, sondern auf dem Windows Kernel mit einigen Anpassungen für den Bedarf mobiler Plattformen [WM13].

Der Systemstart wird bei Windows Phone durch das *Unified Extensible Firmware Interface (UEFI)* abgesichert. Diese Schnittstelle sitzt zwischen dem Betriebssystem und der Firmware. Ein Start des Boot-Prozesses ist dabei nur möglich, wenn dessen Komponenten (z. B. das Betriebssystem) mit einer digitalen Signatur versehen sind, die der *UEFI*-Datenbank bekannt ist [WR13]. Im Anschluss an diese *UEFI*-Überprüfung startet der sogenannte *Trusted Boot*-Schritt. In diesem wird das Betriebssystem selbst überprüft, um sicherstellen zu können, dass dieses nicht von Malware korrumpiert wurde. Schlägt der *Trusted Boot*-Schritt fehl, so kann das System nicht gestartet werden [Mic14].

Um Apps ebenfalls gegen Manipulationen abzusichern und Schadsoftware zu reduzieren, muss auch unter Windows Phone jede App signiert werden. Im Gegensatz zu iOS kann dies allerdings von jedem bei Microsoft registrierten Entwickler durchgeführt werden. Dadurch lässt sich der Ursprung von Schadsoftware zurückverfolgen und Microsoft kann gegebenenfalls Entwicklern das Recht Apps zu signieren wieder entziehen; eine zusätzliche Signatur durch Microsoft ist nicht erforderlich [PW14]. Seit Windows 8 müssen alle Apps, die über den offiziellen *App-Store* veröffentlicht werden zusätzlich von Microsoft selbst signiert werden [Her13]. Im Vergleich zu den umfassenden Tests, die Apple durchführt, kommen diese Maßnahmen allerdings erst zum Einsatz, nachdem Schadsoftware veröffentlicht wurde; Apps, die über andere *App-Stores* verteilt werden (z. B. *Windows Phone AppList*[4]), benötigen die Signatur von Microsoft nicht.

Um den potentiellen Schaden durch Malware daher bereits im Vorfeld einschränken zu können, setzt Microsoft auf ein erweitertes Sandbox-Modell. In diesem sogenannten Kammermodell gibt das System zwei (Windows Phone 8) beziehungsweise vier (Windows Phone 7) Ausführungsumgebungen (die sogenannten Kammern) für Apps vor. Jeder dieser Kammern wird ein Satz

[4] siehe http://www.windowsphoneapplist.com

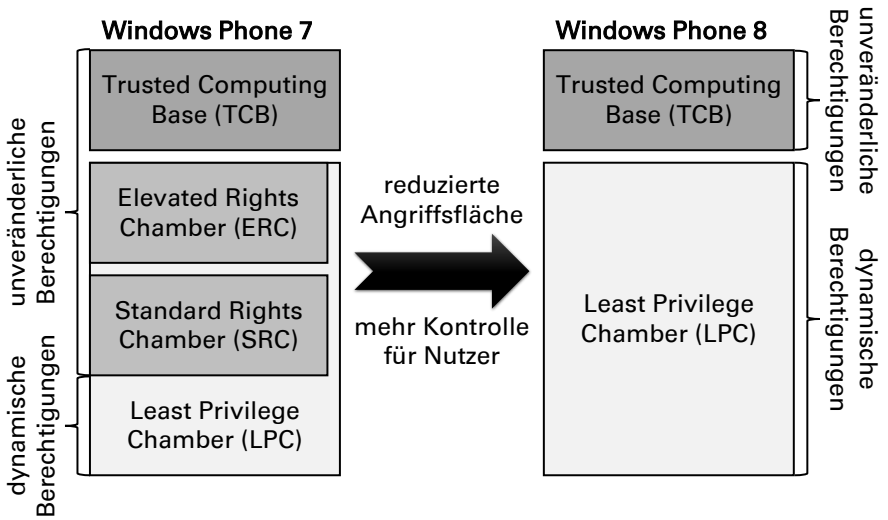


Abbildung 2.2: Das Kammermodell von Windows Phone (in Anlehnung an [Her13; Joy14])

Zugriffsrechte unter Berücksichtigung des *Least-Privilege*-Prinzips[5] zugewiesen. D. h. es muss möglichst feingranular sichergestellt werden, dass eine App nur ein Mindestmaß an Rechten besitzt, die für deren Ausführung notwendig sind [Sch04]. Eine App wird abhängig davon, wie vertrauenswürdig sie ist, in einer der Kammern ausgeführt.

Die *Trusted Computing Base (TCB)*-, *Elevated Rights Chamber (ERC)*- und *Standard Rights Chamber (SRC)*-Kammer haben jeweils eine feststehende Anzahl an Berechtigungen. Da die *TCB*-Kammer uneingeschränkte Zugriffsrechte besitzt, ist diese Systemprozessen vorbehalten. Auch in der *ERC*-Kammer dürfen nur Apps ausgeführt werden, die direkt von Microsoft stammen. Die Apps in dieser Kammer haben ebenfalls uneingeschränkte Zugriffsrechte auf alle Systemressourcen mit Ausnahme der Sicherheitsrichtlinien. Die *SRC*-Kammer ist für Original Equipment Manufacturer (OEM)-Entwickler gedacht, die keine systemweiten Zugriffsrechte benötigen. Apps von Drittanbietern sollten hingegen in der *Least Privileged Chamber (LPC)*-Kammer ausgeführt werden. Diese Apps haben zunächst keinerlei Berechtigungen und der Nutzer muss für jede App

[5] siehe Saltzer und Schroeder [SS75]

selbst entscheiden, welche Berechtigungen dafür angemessen sind [Roo12]. In Windows Phone 8 wurden die *ERC*- sowie *SRC*-Kammer entfernt, um die mögliche Angriffsfläche durch zu viele Berechtigungen zu reduzieren. Alle Apps außer den Systemprozessen müssen dadurch in der *LPC*-Kammer ausgeführt werden [Joy14].

Dieses Kammermodell ist in Abbildung 2.2 schematisch abgebildet. Zur Einschränkung der Zugriffsrechte greift Windows Phone auf Teile des Sicherheitskerns von Windows NT zurück. So kommen u. a. *Access Control Lists (ACLs)* zum Einsatz, mit denen feingranular festgelegt werden kann, welche Zugriffsrechte (sowohl auf Dateien als auch auf Systemdaten) einzelne Benutzer oder, im Falle von Windows Phone, einzelne Apps haben [War98]. Eine App hat dabei keinen freien Zugriff auf das Dateisystem, sondern jeweils nur auf einen isolierten privaten Speicherbereich [Roo12].

Damit Apps auch in der *LPC*-Kammer bei Bedarf auf Sensoren, Systemdaten oder Daten von anderen Apps zugreifen können, muss dies der Nutzer zunächst explizit genehmigen. Hierfür definiert der App-Entwickler, welche Berechtigungen seine App benötigt. Unter Windows Phone spricht man hier von *Capabilities*. Diese *Capabilities* werden dem Nutzer bei der Installation der App angezeigt und er kann entscheiden, ob die angeforderten Berechtigungen der App gewährt werden sollten. Entspricht die Berechtigungsanforderung nach der Meinung des Nutzers nicht dem *Least-Privilege*-Prinzip, so wird die App nicht installiert. Bis einschließlich Windows Phone 7 wurden die angegebenen *Capabilities* noch automatisch auf die tatsächlich benötigten reduziert. Dies wurde mit Windows Phone 8 allerdings geändert. Somit muss der Nutzer alle vom Entwickler angegebenen Berechtigungen akzeptieren, auch wenn diese für die App eigentlich gar nicht benötigt werden [WM13].

Windows Phone bietet standardmäßig keine vollständige Verschlüsselung des Speichers an, um die Daten gegen physische Angriffe zu verteidigen. Hierfür muss der Nutzer zunächst für spezielle Sicherheitsfeatures freigeschaltet werden und er muss Sicherheitsrichtlinien definieren. Für diesen Zweck gibt es keine grafische Benutzeroberfläche und die Konfiguration erfolgt vollständig über *Exchange ActiveSync (EAS)*[6]. Auf diese Weise werden viele Nutzer ohne

[6] Mittels *EAS* können Zertifikate oder Sicherheitsrichtlinien an mobile Geräte ausgeliefert werden. Aufgrund des hierfür benötigten Back-Ends richtet sich *EAS* in erster Linie an Unternehmen und nicht an Privatkunden [Jac16].

profunde IT-Kenntnisse davon abgehalten die Datenverschlüsselung zu nutzen. Daher sind App-Entwickler dazu angehalten selbstständig für den Schutz der Daten ihrer App zu sorgen [PW14]. Die Verschlüsselung selbst basiert auf Windows *BitLocker*, einer Architektur zur zuverlässigen und sicheren Verschlüsselung von Datenträgern [Fer06]. Dabei werden Prozessoren unterstützt, die ein *Trusted Platform Module (TPM)*[7] besitzen. Allerdings ist ein solcher Prozessor keine Anforderung seitens Windows Phone [Mic14]. Die Sicherheit des Verschlüsselungsalgorithmus ist demnach abhängig von der verwendeten Hardware unterschiedlich [ACG15].

Ein weiterer Schutz gegen physische Angreifer stellt das Sperren des Telefons dar. Dies wird auch von Windows Phone unterstützt, allerdings bietet Microsoft von Haus aus nur simple Sicherheitsvarianten an, wie beispielsweise die Eingabe einer einfachen PIN. Tiefgreifendere Sicherheitsrichtlinien müssen ebenfalls von IT-Experten mittels *Exchange ActiveSync* aufgespielt werden [WM13]. Auf diese Weise kann u. a. eine Mindestlänge für den Entsperrcode, eine Maximalzahl an zulässigen Fehlversuchen oder ein Ablaufdatum für den Entsperrcode festgelegt werden [WP14].

Windows Phone war von Anfang an nicht auf einen einzigen Hardwarehersteller ausgerichtet und öffnet sich immer mehr Hardwareplattformen. Daher kann auch bei der Sicherheit keine spezielle Hardware vorausgesetzt werden, wie es bei Apple der Fall ist, sondern die Software muss zu unterschiedlichen Plattformen kompatibel sein. Dies schlägt sich vor allem auf die Verschlüsselung nieder. Zusätzlich ist das Sicherheitsmodell stark auf Unternehmen ausgelegt. Für Privatkunden ist das Sicherheitssystem soweit vereinfacht, dass es zwar einfach nutzbar ist, aber viel Verantwortung auf die Apps bzw. deren Entwickler abgeschoben wird. Das Kammermodell nach dem *Least-Privilege*-Prinzip ist zwar sehr leistungsstark und grundsätzlich sicher, aber aufgrund der Alles-oder-Nichts-Entscheidungen, die der Nutzer bei der Rechtevergabe treffen muss, erhalten Apps dennoch häufig zu viele Berechtigungen. Wie Getzmann und Nowak [GN14] allerdings herausstellen, ist Windows Phone für den Einsatz im Unternehmensumfeld konzipiert und hierfür bietet es Dank der Features, die durch *EAS* ermöglicht werden, ein überzeugendes Sicherheitskonzept. Darüber hinaus erweitert und überarbeitet Microsoft fortwährend

[7] siehe Morris [Mor11]

grundlegende Elemente des Sicherheitskonzepts, um so die Datensicherheit zu erhöhen [Ols12; Ols14].

2.3 Android von Google

Auch Google setzt bei Android auf ein mehrstufiges Sicherheitssystem. Allerdings gibt es für Android keine dedizierte Hardware, wodurch die Sicherungsmaßnahmen rein softwareseitig ausgeführt werden. Dadurch sind umfassende Schutzmechanismen, wie beispielsweise eine hardwareunterstützte Verschlüsselung der Daten, unter Android nicht möglich. Nahezu alle Mechanismen beruhen dabei auf Linux-Diensten, die allerdings auf die Bedürfnisse einer mobilen Plattform angepasst wurden [WM12b]. Dabei werden mit jeder neuen Android-Version weitere Sicherheitsfeatures hinzugefügt. Während die ersten Android-Versionen nahezu keinerlei Schutzmechanismen besaßen, konnten so nach und nach mehrere Schwachstellen geschlossen werden. Aktuell ist Android allerdings noch weit davon entfernt ein umfassendes Sicherheitssystem zu bieten [SPDS14], wie die folgende Übersicht über die vorhandenen Sicherungsmaßnahmen zeigt.

Erst seit Android 4.4 wird der Boot-Prozess abgesichert. Hierzu wird *dm-verity*[8] genutzt. Durch dieses Kernel-Feature kann die Integrität des Dateisystems überprüft werden und es kann sichergestellt werden, dass keine Schadsoftware Manipulationen an Systemdaten durchgeführt hat [ZAH+14]. Da diese Überprüfung vom Kernel durchgeführt wird, muss beim Systemstart zusätzlich sichergestellt werden, dass der Kernel nicht manipuliert wurde. Der Integritätstest des Kernels ist allerdings nicht standardisiert und muss vom jeweiligen Gerätehersteller realisiert werden [Ele14].

Eine systematische Überprüfung von Apps, die von Drittanbieter stammen, findet bei Android nicht statt. Dies würde der liberalen Philosophie hinter Android widersprechen, da es eine Einschränkung für App-Entwickler darstellen würde [AJ10]. Erst wenn eine App Google als Schadsoftware gemeldet wurde, wird die App gesperrt, allerdings nur für den offiziellen *App-Store* [Man12]. Auch werden App-Entwickler nicht registriert. Jeder darf Apps für Android entwickeln und veröffentlichen. Die erforderliche Signatur kann vom Entwickler

[8] siehe https://gitlab.com/cryptsetup/cryptsetup/wikis/DMVerity

selbst erzeugt werden und dient lediglich dazu, dass Apps einem Ursprung zugeordnet und Apps des gleichen Ursprungs identifiziert werden können. Einzig Systemanwendungen bilden eine Ausnahme, da diese vom Plattformanbieter signiert werden müssen, um umfassendere Berechtigungen zu erhalten [Ele14].

Linux muss als klassisches Mehrbenutzersystem in der Lage sein, die Daten und Prozesse mehrerer Nutzer parallel, aber dennoch isoliert zu verwalten. Da Android einen Linux Kernel nutzt, ist auch Android von Grund auf ein Mehrbenutzersystem. Allerdings wurde unter der Annahme, dass Smart Devices in der Regel nur von einem einzigen Nutzer verwendet werden, die maximale Anzahl an Nutzern softwareseitig auf Eins gesetzt. Dennoch kommt das Linux Mehrbenutzerkonzept Android zu Gute. Statt Nutzern mit eindeutigen Identifikatoren (*User Identifiers (UIDs)*) auszustatten, werden diese an Apps vergeben und jede *UID* wird in einem isolierten Prozess ausgeführt. Auch besitzt jede *UID* einen dedizierten Speicherbereich, auf den nur sie Zugriff hat. Auf diese Weise wird jeder Prozess in einer isolierten Sandbox ausgeführt. Apps mit dem gleichen Entwicklerzertifikat können sich eine *UID* teilen und somit in einer gemeinsamen Sandbox ausgeführt werden. Dadurch sind sie in der Lage, Daten untereinander auszutauschen [BCMvO12].

Dieses Sandbox-Modell basiert auf *Discretionary Access Control (DAC)*. *DAC* weist einer App Berechtigungen zu, mit denen diese machen kann, was sie möchte. Dies beinhaltet u. a. die unkontrollierte Weitergabe der Berechtigung an andere *UIDs*. Um dem entgegenwirken zu können, setzt Android *SEAndroid*[9] ein, eine angepasste Version von *SELinux*[10]. Mit *SEAndroid* können gemäß des *Mandatory Access Control (MAC)*-Modells systemweite Richtlinien definiert werden, die für alle Nutzer, d. h. im Fall von Android für alle Apps gelten. So kann beispielsweise die Weitergabe von bestimmten Berechtigungen untersagt werden. Allerdings kontrolliert *SEAndroid* nur Systemdienste mit besonders weitreichenden Privilegien. Bei allen anderen Apps werden Verstöße gegen die Richtlinien nur registriert und protokolliert, nicht aber verhindert [SC13b].

Damit eine App dennoch mit Apps anderer Entwickler oder Systemanwendungen oder -diensten außerhalb ihrer Sandbox kommunizieren kann, stellt auch Android APIs zur Verfügung. Wie in Abbildung 2.3 zu sehen ist, haben

[9] siehe http://seandroid.bitbucket.org/
[10] siehe https://selinuxproject.org/

Apps nur direkten Zugriff auf das Anwendungsframework. In diesem befinden sich APIs zu allen Sensoren und Systemdiensten, über die Apps kontrolliert auf Daten zugreifen können. Die Kommunikation erfolgt über *Binder*, einem *Inter-Process Communication (IPC)*-Mechanismus speziell für Android. Mit den *Bindern* wird der Datenaustausch über Prozessgrenzen hinweg ermöglicht. Allerdings ist das Hauptaugenmerk bei den *Bindern* auf die Flexibilität gerichtet, weshalb auf viele Techniken, die die Sicherheit eines Systems erhöhen, bewusst verzichtet wurde, wie beispielsweise Semaphore für die geteilte Nutzung beschränkter Ressourcen. Nur die Systemdienste haben über die Hardwareabstraktionsschicht direkten Zugriff auf die Sensoren. Diese Abstraktionsschicht gestattet es Android auf divergierenden Geräten lauffähig zu sein [Ele14].

In dem Anwendungsframework von Android befinden sich auch die *Content Provider*. Ein *Content Provider* stellt eine Schnittstelle zwischen Apps dar. Diese Schnittstelle wird von der App, die ihre Daten freigeben möchte, definiert, und jede andere App kann über den *Content Provider* bzw. dessen Uniform Resource Identifier (URI) darauf zugreifen, ohne dass der Besitzer der Daten dies in irgendeiner Weise reglementieren kann [CFGW11].

Android-Apps sind in zwei unterschiedlichen Partitionen gespeichert. In einer Partition liegen die Systemanwendungen. Dies beinhaltet alle Apps, die vom Plattformanbieter signiert wurden. Für Apps in dieser Partition gelten besondere Rechte. Einerseits können sie vom Nutzer nicht gelöscht werden und andererseits werden ihnen mehr Berechtigungen zugestanden. Da jeder Plattformanbieter selbst entscheiden kann, mit welchen Apps er sein System ausliefern möchte, finden sich hier auch Apps von Drittanbietern (z. B. die *Facebook*-App), die dadurch ebenfalls automatisch mehr Rechte haben. In der anderen Partition befinden sich alle Daten des Nutzers und auch alle von ihm installierten Apps [Hoo12].

Android unterstützt seit Version 3.0 Full-disk Encryption (FDE). Während FDE eigentlich dafür steht, dass außer einem kleinen Bereich, der für die Aufrechterhaltung des Betriebs nötig ist, sämtliche Daten verschlüsselt werden, beschränkt sich Android dabei auf die Nutzerdatenpartition. Die Boot- sowie die Systempartition und damit das Betriebssystem selbst bleiben davon unberührt, wodurch diese durch das FDE nicht geschützt sind [Ele14]. Und obwohl die Verschlüsselungsalgorithmen ständig verbessert werden [And16k], ist Android

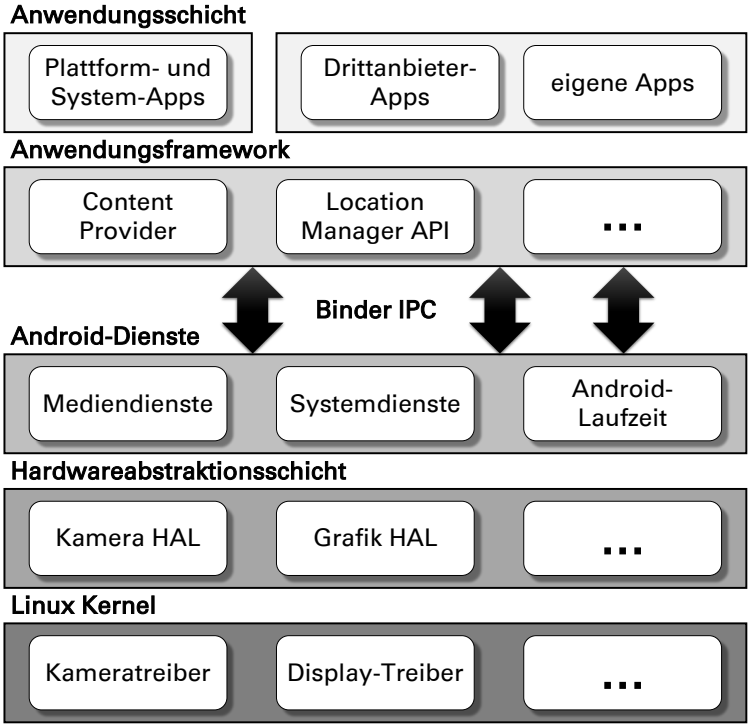


Abbildung 2.3: Die Android-Systemarchitektur (in Anlehnung an [And16j; BP15; Ele14])

bezüglich Sicherheit und Zuverlässigkeit weit hinter den hardwaregestützten Verfahren von iOS zurück [Goo16].

Die Verschlüsselung gilt nur so lange, wie das Gerät vom Nutzer gesperrt ist [GM14]. Zum Entsperren bietet Android mehrere unterschiedliche Verfahren an, angefangen bei der Eingabe einer PIN, über die Eingabe eines Musters bis hin zur Gesichtserkennung. Dies macht deutlich, dass die Verschlüsselung nur gegen Angreifer von außen, nicht aber gegen Schadsoftware schützen soll [SS12a].

Aber auch gegen den Datendiebstahl oder -missbrauch gibt Android dem Nutzer Mechanismen an die Hand. Im Gegensatz zu iOS kann der Nutzer unter Android theoretisch selbst entscheiden, welche Berechtigungen eine

App besitzt. Für die Funktionen des Anwendungsframeworks legt das System fest, welche Berechtigungen nötig sind, damit die jeweilige Funktion verwendet werden kann. Eine Berechtigung ist unter Android als *Permission*, einer textuellen Repräsentation der Berechtigung, abgebildet. Jede App definiert einen Satz *Permissions*, die sie zur Ausführung benötigt. Wird eine Funktion aufgerufen, die eine nicht deklarierte *Permission* benötigt, so wirft das System eine Sicherheits-Exception. App-Entwickler können weitere Berechtigungen definieren, die andere Apps benötigen, um mit der App des Entwicklers interagieren zu können, z. B. mittels *Content Providern*. Bei der Installation der App bekommt der Nutzer die angeforderten *Permissions* angezeigt und kann entscheiden, ob er der App die dazugehörigen Berechtigungen erteilen will. Allerdings handelt es sich dabei um eine Alles-oder-Nichts-Entscheidung, d. h. der Nutzer muss alle *Permissions* genehmigen, um die App installieren und damit ausführen zu können. Auch können nachträglich keine Berechtigungen entzogen werden [All15].

Mit Android 6.0 führte Google daher sogenannte *Runtime Permissions* ein. Diese *Permissions* kann der Nutzer auch zur Laufzeit einer App gewähren oder wieder entziehen [BJM+16]. Allerdings gilt dies nicht für alle *Permissions* [And16h]. Mit den *Runtime Permissions* erhöhen sich augenscheinlich die Einflussmöglichkeiten, die einem Nutzer gegeben werden, um die Berechtigungen einer App zu regulieren. Allerdings werden unter einer *Permission* teilweise eine Vielzahl an Berechtigungen zusammengefasst, wodurch die Benutzerfreundlichkeit dieser Art der Berechtigungsvergabe nicht zufriedenstellend ist [MMK+16]. Darüber hinaus wird ein Nutzer durch die hohe Frequenz, mit der z. T. gefährliche *Permissions* angefordert werden, derart desensibilisiert, dass er diesen oft blind zustimmt [FEW12], obwohl er einer App diese Berechtigung eigentlich gar nicht hätte erteilen wollen [Bao16].

Da das *Permission*-System aber eine zentrale Rolle im Android-Sicherheitssystem darstellt, stellt dieses die Schutzziele [IS1] (*Vertraulichkeit*), [IS2] (*Integrität*), [IS5] (*Nachvollziehbarkeit*), [IS6] (*Authentizität*) und [IS7] (*Verbindlichkeit*) nur dann sicher, wenn jede App nur die Berechtigungen besitzt, die ihr vom Nutzer zugeteilt wurden. Wie Untersuchungen von Orthacker et al. [OTK+12] allerdings zeigen, können Berechtigungen unter Apps weitergegeben werden, wodurch für Android nicht mehr ersichtlich ist, welche App Daten

anfordert und wohin die Daten weitergeleitet werden. Das Schutzziel [IS3] (*Verfügbarkeit*) wird mit den *Runtime Permissions* erheblich eingeschränkt, sollte eine App die Möglichkeit fehlender Berechtigungen nicht berücksichtigt haben. Da eine App eine Berechtigung entweder besitzt oder nicht und es dabei keine Abstufungen gibt, ist das Schutzziel [IS4] (*Genauigkeit*) nicht gefährdet.

2.4 Vergleich der Plattformen bezüglich Privatheits- und Sicherheitsaspekte

Barrera und Van Oorschot [BV11] stellen in ihrer Arbeit einen Vergleichskatalog von Sicherheitsaspekten für mobile Plattformen auf. Dabei heben sie die folgenden sechs Eigenschaften als besonders charakteristisch hervor: Der Level auf dem die *Isolation* stattfindet, der Umgang mit digitalen *Codesignaturen*, die Möglichkeit für einen Nutzer *Root*-Rechte zu erlangen, das Vorhandensein eines *Kill Switches*, die unterstützten *App-Bezugsquellen* sowie das zugrundeliegende *Sicherheitsmodell*. Hogben und Dekker [HD10] erweitern diese Liste der Eigenschaften um den Aspekt des verwendeten *Berechtigungsmodells*. Im Folgenden werden die drei mobilen Plattformen iOS, Windows Phone und Android bezüglich dieser sieben Kriterien verglichen.

Isolationslevel. Bei nahezu allen mobilen Plattformen kommt eine Art Isolationstechnik zum Einsatz, damit sich parallel laufende Apps nicht gegenseitig beeinflussen können. Bei mobilen Plattformen wird zusätzlich darauf geachtet, dass eine App nicht auf die Daten anderer Apps zugreifen kann [BV11]. Dadurch kann die Bedrohung, die durch potentielle Schadsoftware ausgeht gemindert werden. Allerdings unterscheiden sich die drei untersuchten Plattformen bezüglich des Levels der Isolation, d. h. wie strikt Apps voneinander getrennt ausgeführt werden.

Unter iOS wird jede App in einer eigenen Sandbox ausgeführt. Diese Sandbox isoliert jedoch nicht nur Apps untereinander, sondern auch von den Systemressourcen, wie dem Dateisystem oder dem Netzwerk [AMN+13]. Für jede App und jede Sandbox kann das System separat feingranulare Regeln festlegen, welche Zugriffe aus der Sandbox heraus zulässig sind. Der Nutzer hat allerdings keinen Einfluss auf diese Zugriffsregeln [PS11]. Windows Phone verwendet eine wesentlich grobgranularere Isolationstechnik. Es wird ein Kammermodell

eingeführt, in dem jede Kammer unterschiedlich viele Rechte besitzt mit dem System, Apps oder Dateien zu interagieren. Kritische Apps werden vom System in weniger privilegierten Kammern ausgeführt. Innerhalb einer Kammer gelten weniger strikte Isolationskriterien, d. h. Apps, die in der gleichen Kammer laufen, können miteinander kommunizieren [MTDG11]. Android isoliert Apps feingranularer bereits auf Prozessebene. Allerdings können mehrere Apps im selben Prozess ausgeführt werden und somit auf dem gleichen Speicher arbeiten, wenn die Apps von dem gleichen Entwickler stammen [SFK+10].

Codesignatur. Software wird heutzutage kaum noch auf physischen Datenträgern verkauft, sondern als Downloads angeboten werden. Der Nutzer muss darauf vertrauen, dass die Software unverändert zu ihm gelangt. Kriminelle haben sich aber gerade im Sektor der Smart Devices darauf spezialisiert, dass sie Schadcode in bekannte Apps einbauen und so heimlich ihre Schadsoftware über diese Apps verbreiten. Um der Manipulation entgegenzutreten, kann ein Entwickler seine App mittels Public-Private-Key-Verfahren[11] digital signieren [Geb12]. Dabei lassen sich zwei Vorgehensweisen unterscheiden: Entweder signiert der Entwickler seine App selbst, was eine schnelle Veröffentlichung ermöglicht, oder er gibt die App an eine vertrauenswürdige dritte Instanz, die die Identität des Entwicklers sowie die App selbst prüft und mit der Signatur diese akkreditiert [Sch10].

Jede App für iOS muss von Apple signiert werden, um lauffähig zu sein. Hierzu muss der Entwickler seine App Apple zur Überprüfung überlassen. Nach welchen Kriterien Apple eine App untersucht, ist jedoch nicht bekannt. Es gilt allerdings als erwiesen, dass neben automatischen Tests auf Schadsoftware auch manuelle Begutachtungen durchgeführt werden, um zu prüfen, ob die App inhaltlich den Richtlinien von Apple entspricht. Die digitale Signatur dient bei Apple zur Kontrolle, dass unter iOS nur sichere Apps, d. h. Apps, die Apple technisch und inhaltlich unbedenklich erscheinen, lauffähig sind [BV11; MDTG11]. Dieses Vorgehen ist sehr zeitintensiv, wodurch es mitunter lange Zeit dauert, bevor eine App freigegeben wird. Außerdem birgt es Gefahren, da es keinen vollständigen Schutz gegen Schadsoftware bietet, dies dem Nutzer jedoch suggeriert wird. Dieser muss dem Urteil von Apple blind vertrauen [Mil11]. Windows Phone ist weniger restriktiv, da Entwickler von Microsoft ein digi-

[11] siehe Rivest, Shamir und Adleman [RSA78]

tales Zertifikat anfordern können und mit diesem ihre Apps selbst signieren können[12]. Die Signatur entscheidet abhängig davon, wie vertrauenswürdig der Entwickler ist, in welcher Kammer eine App ausgeführt werden darf und ob sie vom System zur Laufzeit noch weiter überprüft werden muss. Darüber hinaus dient die Signatur auch als digitaler Fingerabdruck des Entwicklers, wodurch sich Schadsoftware eindeutig zurückverfolgen lässt [MDTG11]. Android ist hinsichtlich der digitalen Signatur die liberalste mobile Plattform, da sich Entwickler selbstständig Zertifikate ausstellen und ihre Apps damit signieren können. Die Signatur dient in diesem Fall nur dem Schutz der App gegen Veränderung und nicht dem Schutz des Nutzers [BV11].

Berechtigungsmodell. Da auf Smart Devices bedingt durch die hohe Anzahl an Sensoren und die mannigfaltigen Verwendungszwecke sehr viele private Daten des Nutzers anfallen, muss sichergestellt werden, dass Apps nur auf solche Daten Zugriff bekommen, die auch für sie bestimmt sind. Häufig ist es völlig legitim und für die Funktionalität der App unabdingbar, dass sie auf private Daten zugreifen darf. Dennoch muss in einem Berechtigungsmodell möglichst genau festgelegt werden, welche Berechtigungen der App eingeräumt werden sollen, um einen unkontrollierten Datenzugriff zu verhindern. Da nur der Nutzer endgültig entscheiden kann, wie viel Privatheit er bereit ist für die Services einer App preiszugeben, sollte er bei der Berechtigungsvergabe maßgeblich mit eingebunden werden [HD10].

Apple verwendet kein dediziertes Berechtigungsmodell und jede iOS-App besitzt grundsätzlich uneingeschränkte Rechte. Jeder Entwickler muss bestätigen, dass sich seine App nicht böswillig im Umgang mit den Daten des Nutzer verhält und diesen insbesondere darüber informieren, wenn seine Daten an Dritte weitergeleitet werden. Außerdem darf eine App nur auf die Daten zugreifen, die für die Ausführung der App erforderlich sind. Da es technisch unter iOS keinerlei Kontrollmechanismen gibt, die die Umsetzung dieser Vereinbarung durchsetzen können, fußt das Berechtigungsmodell darauf, dass Apple jede App signieren muss, damit diese lauffähig ist. Somit kann Apple jede App vor der Veröffentlichung daraufhin überprüfen, ob die App den vereinbarten Nutzungsbedingungen entspricht. Da allerdings trotz dieser Kontrollen immer wieder Schadsoftware in *iTunes* auftaucht und der Nutzer keinerlei Möglichkei-

[12] Seit Windows Phone 8 müssen alle Apps zusätzlich von Microsoft signiert werden [Her13].

ten hat, die Rechte einer App einzuschränken, ist dieses Berechtigungsmodell als kritisch einzustufen [EKKV11].

Unter Windows Phone gibt es zwei unterschiedliche Berechtigungsmodelle, von denen je nach Gerätetyp eines zum Einsatz kommt. Im sogenannten *One-Tier-Access-Modell* wird lediglich festgelegt, ob eine App ausgeführt werden darf, ohne dabei deren Datenverwendung zu berücksichtigen. Wurde die App von einem vertrauenswürdigen Entwickler signiert, so bekommt sie in diesem Modell vollen Zugriff auf die APIs aller Systemfunktionen. Bei allen anderen Apps muss der Nutzer der Ausführung der App zunächst zustimmen. Gestattet er die Ausführung, so bekommen allerdings auch diese Apps vollen Zugriff auf die APIs. Im *Two-Tier-Access-Modell* wird zusätzlich für Apps ohne vertrauenswürdige Signatur überprüft, auf welche APIs eine App zur Laufzeit zugreift. Abhängig von dieser Überprüfung wird die App mit eingeschränkten Rechten gestartet [MDTG11]. Der Nutzer hat auf die Berechtigungen, die eine App bekommt, in beiden Modellen wenig bis keinen Einfluss. Autorisiert er die Ausführung einer App, so erhält diese die Berechtigungen, die Microsoft abhängig von der Signatur angemessen findet bzw. die von der App angefordert werden.

Android setzt auf ein nutzerzentrisches Berechtigungsmodell, in dem Apps Berechtigungen (die sogenannten *Permissions*) explizit anfordern müssen, bevor sie über APIs auf private Daten oder kritische Systemfunktionen Zugriff bekommen. Eine Prüfung, ob die angeforderten Berechtigungen angemessen sind, muss der Nutzer durchführen, da seitens des Systems keine weiteren Schutzmaßnahmen getroffen werden. Alle *Permissions* einer App werden bei der Installation angezeigt[13] und der Nutzer muss alle genehmigen, um die Installation fortführen zu können [EFW13]. Da das Geschäftsmodell vieler kostenloser Apps allerdings darauf beruht, möglichst viele private Daten eines Nutzers für Werbezwecke zu gebrauchen [LEPM12] und das Verhalten der Nutzer im Umgang mit Apps zu analysieren, ist die Liste der angeforderten *Permissions* sehr umfangreich [EOMC11]. Obwohl bei diesem Berechtigungsmodell die Kontrolle in der Hand des Nutzers liegt, muss er de facto allen *Permissions* zustimmen, um eine App nutzen zu können. Erschwerend kommt

[13] Seit Android 6.0 ist es für einige wenige Permissions möglich, diese einer App erst zur Laufzeit zu genehmigen bzw. wieder zu entziehen (siehe Abschnitt 2.3).

hinzu, dass viele Nutzer mit den grobgranularen Entscheidungsmöglichkeiten in Android schlicht überfordert sind [FHE+12; FEW12].

Root-Rechte. Der Begriff *Root*-Rechte bezeichnet, dass spezielle Nutzer vollen Zugriff auf alle Systemressourcen und -funktionen besitzen, während alle anderen Nutzer nur eingeschränkte Rechte besitzen [Got12]. Viele mobile Plattformen nutzen diese Technik, um zwischen einer unsicheren Ausführungsumgebung für Apps von Drittanbietern und einer sicheren Ausführungsumgebung für Systemanwendungen unterscheiden zu können. Häufig sind auch die Speicherbereiche der beiden App-Typen strikt voneinander getrennt, um eine Kompromittierung des Systems zu verhindern [BV11].

Alle drei untersuchten mobilen Plattformen besitzen einen geschützten Speicherbereich für das Betriebssystem, und dem Nutzer wird nicht ermöglicht *Root*-Rechte zu erlangen [PS11]. Für viele Apps (u. a. Virenscanner oder Backup-Programme) benötigt der Nutzer allerdings einen Vollzugriff auf das System [MK97]. Daher kommen immer mehr Werkzeuge zum Einsatz, die durch die Ausnutzung von Schwachstellen des Systems dem Nutzer *Root*-Rechte verschaffen [SCB15]. Diese Werkzeuge können allerdings selbst Schadsoftware beinhalten [YN11] oder den Erfolg anderer Schadsoftware begünstigen [Lea11]. Da sich die Hersteller dieser Gefahr bewusst sind, unterstützen einige Hersteller von Android-basierten Smart Devices den Nutzer dabei *Root*-Rechte zu erlangen, um diese Gefährdung zu minimieren [Son16].

Kill Switch. Unter einem *Kill Switch* versteht man im Umfeld von mobilen Plattformen eine Technik, die es den Anbietern der mobilen Plattform gestattet den Download von Apps zu sperren, wenn diese als Schadsoftware identifiziert wurden. Darüber hinaus kann darüber auch einen Löschbefehl an alle Geräte geschickt und so die betreffende App gelöscht werden, ohne physischen Zugriff auf die Geräte zu erlangen. Auf die gleiche Weise können Apps auch automatisch ersetzt werden, z. B. wenn es ein Sicherheitsupdate für die App gibt. Der Nutzer kann weder die Löschung noch die Ersetzung einer App verhindern [BEvO11].

Alle drei untersuchten mobilen Plattformen beinhalten einen Kill Switch, mit dem sie Apps per Fernzugriff löschen können. Allerdings sind darüber hinaus nur wenige Details bezüglich der Verwendungshäufigkeit und den Gründen, warum Apps gelöscht werden, öffentlich [ME10; MDTG12].

App-Bezugsquellen. Nahezu jeder Anbieter einer mobilen Plattform betreibt auch einen *App-Store*, über den Nutzer Apps von Drittanbietern für die Plattform beziehen können. Dabei beeinflusst die Qualität und Quantität der über den *App-Store* verfügbaren Apps unmittelbar die Verkaufszahlen von mobilen Geräten, auf denen die jeweilige Plattform läuft [Kim10]. Allerdings unterscheiden sich die Strategien der Plattformanbieter im Umgang mit dem *App-Store* stark. Gemäß Gonçalves, Walravens und Ballon [GWB10] lassen sich im Umfeld mobiler Plattformen zwei Modelle unterscheiden, das *Enabler-Modell* und das *Systemintegrator-Modell*. Im Enabler-Modell steht Anbietern von Drittsoftware eine Vertriebsplattform zur Verfügung, sie sind aber nicht an diesen Vertriebsweg gebunden. Im Systemintegrator-Modell läuft im Gegensatz dazu der vollständige Vertrieb von Apps einzig über den Plattformanbieter.

Apple folgt dem Systemintegrator-Modell, d. h. unter iOS können Apps nur über den offiziellen *App-Store iTunes*[14] bezogen werden. Dadurch kontrolliert Apple das Angebot an verfügbaren Apps für iOS. Untersuchungen zeigen, dass dieses restriktive Vorgehen, das Vorkommen von Malware erheblich reduziert [FFC+11]. Microsoft betreibt mit dem *Microsoft Store* ebenfalls einen einzigen offiziellen *App-Store*, allerdings können sich Nutzer das *Sideloading*, d. h. den Bezug von Apps über andere Quellen, von Microsoft freischalten lassen [KNRR16]. Android folgt vollständig dem Enabler-Modell, da es mit *Google Play* zwar einen offiziellen *App-Store* besitzt, aber auch beliebige andere App-Bezugsquellen genutzt werden können. Da vor allem Apps aus alternativen Quellen nahezu ungeprüft sind, verbreitet sich Malware hier besonders stark [ZWZJ12].

Tabelle 2.1 gibt einen Überblick über die wesentlichen Erkenntnisse des Vergleichs der drei mobilen Plattformen. Das Bewertungskriterium des verwendeten **Sicherheitsmodells** nimmt eine Sonderstellung ein, da hierbei die restlichen Kriterien Einfluss nehmen. Daher wird dieses Kriterium gesondert in Abschnitt 2.5 betrachtet.

Wie aus diesem Vergleich hervorgeht, geht Android am liberalsten mit Berechtigungen für Apps um. Die Kontrolle über die Berechtigungen und somit darüber, auf wie viele und welche privaten Daten eine App Zugriff erhält,

[14] *iTunes* wird im Rahmen dieser Arbeit als Sammelbegriff für alle digitalen Vertriebsplattformen von Apple, wie den *Apple App Store* oder den *iTunes Store*, verwendet, da diese auf die Infrastruktur von *iTunes* zurückgreifen.

Feature	**iOS**	**Windows Phone**	**Android**
Isolation	systemweit	Kammermodell	prozessbasiert
Codesignatur	Überwachung	Identifikation	App-Schutz
Berechtigungen	Apple	Signatur	Nutzer
Root-Rechte	Nein	Nein	Nein*
Kill Switch	Ja	Ja	Ja
App-Quelle	*iTunes*	*Microsoft Store*†	nicht proprietär
Sicherheitsmodell	Walled Garden	Guardian	User Control

* Viele Android-Derivate gestatten es dem Nutzer *Root*-Rechte zu erhalten.
† Sideloading kann unter Windows Phone aktiviert werden.

Tabelle 2.1: Vergleich der heute vorherrschenden Mobilplattformen

liegt vollständig beim Nutzer. Dies realisiert weitestgehend die Anforderung [DP1] (*Einwilligung*) und teilweise die Anforderung [DP6] (*Kontrolle*) an ein Datenschutzsystem. Jedoch ist die Kontrolle dadurch eingeschränkt, dass eine Ablehnung einer angeforderten Berechtigung zur Folge hat, dass die betroffene App nicht installiert werden kann. Auch hat der Nutzer keinerlei Einfluss auf die Anforderungen [DP2] (*Zweckbindung*), [DP3] (*Erforderlichkeit*) sowie [DP5] (*Datensparsamkeit*). Wie die Untersuchungen von Felt et al. [FHE+12] darüber hinaus zeigen, ist das *Permission*-System für die meisten Nutzer zu kompliziert, um nachvollziehen zu können, welche Konsequenzen mit den *Permissions* verbunden sind. Daher ist in diesem System auch die Anforderung [DP4] (*Transparenz*) nicht gegeben.

2.5 Klassifikation von Sicherheitsmodellen

Gemäß Barrera und Van Oorschot [BV11] lassen sich drei grundsätzlich unterschiedliche Sicherheitsmodelle im Umgang mit Apps unterscheiden, die in heutigen mobilen Plattformen zum Einsatz kommen:

Das Walled-Garden-Modell.

Im *Walled-Garden-Modell* wird die mobile Plattform vollständig nach Außen hin abgeschirmt. Apps von Drittanbietern können nur installiert und genutzt werden, wenn dies vom Plattformanbieter gestattet wird. Dem Nutzer stehen demnach nur solche Apps zur Verfügung, die den Richtlinien des Plattformanbieters entsprechen. Apps können vom Plattformanbieter auch nachträglich als unzulässig erklärt werden. Dies bedeutet für den Nutzer nicht nur, dass sie infolgedessen nicht mehr installiert werden können, sondern auch bereits installierte Versionen der App können vom Plattformanbieter deinstalliert werden (via Kill Switch). Auch können ohne das Zutun des Nutzers neue Apps aufgespielt werden, wenn dies vom Plattformanbieter als notwendig erachtet wird.

In diesem Modell ist eine unverfälschliche digitale Signatur von Apps unverzichtbar. Dabei vergibt ausschließlich der Plattformanbieter an Entwickler das Recht Apps zu signieren. Durch die Signatur kann eine App eindeutig identifiziert werden und es wird sichergestellt, dass der Code der App nicht nachträglich verändert wurde.

Der Nutzer muss im Walled-Garden-Modell darauf vertrauen, dass der Plattformanbieter im Interesse des Nutzers handelt, da sämtliche sicherheitsrelevante Entscheidungen vom Anbieter getroffen werden. Ist der Anbieter zuverlässig und geht bei der Prüfung der Apps gewissenhaft vor, so ist das Walled-Garden-Modell selbst für unerfahrene Nutzer sehr sicher. Dennoch sollte die unumschränkte Kontrolle über die Apps (und damit über die Daten der Nutzer), die dem Plattformanbieter in diesem Modell gewährt wird, hinterfragt werden.

Das Guardian-Modell.

Das *Guardian-Modell* ist weniger präskriptiv als das Walled-Garden-Modell. Hierbei überträgt der Plattformanbieter nahezu sämtliche sicherheitsrelevante Entscheidungen an eine Kontrollinstanz, den sogenannten *Guardian*. Der Guardian ist eine sachkundige dritte Instanz. Dies kann beispielsweise ein Servicepartner des Plattformanbieters (wodurch es dem Walled-Garden-Modell sehr nahe kommt) oder ein Sicherheitsexperte,

dem der Nutzer vertraut (z. B. ein Administrator) sein. Auch kann die Entscheidungsgewalt unter mehreren Instanzen aufgeteilt werden.

Während der Guardian die fundamentalsten sicherheitsrelevanten Entscheidungen selbstständig trifft (z. B. welche Berechtigungen eine App erhalten soll), wird der Nutzer bei weniger sicherheitskritischen Fragen in den Entscheidungsprozess miteinbezogen. Hierbei kann der Guardian ihn als Ratgeber unterstützen, z. B. indem er Apps überprüft und vor potentiellen Gefahren warnt. Bei einem größeren Gefahrenpotenzial kann der Guardian die Entscheidung des Nutzers allerdings überstimmen, da dieser aufgrund fehlender Kompetenz sowie mangelnder Sorgfalt als unkalkulierbares Sicherheitsrisiko angesehen wird.

Das User-Control-Modell.

Den beiden obigen Modellen, die den Nutzer mehr oder weniger entmündigen, steht das *User-Control-Modell* gegenüber. In diesem Modell wird dem Nutzer die vollständige Kontrolle über das System, aber auch die komplette Verantwortung über sicherheitsrelevante Entscheidungen übertragen. Drittanbieter sind in diesem Modell nahezu keinen Einschränkungen unterworfen, und der Plattformanbieter führt keine Überprüfung der verfügbaren Apps durch.

In diesem Modell werden weitreichende Sicherheitssysteme dringend benötigt, da Schadsoftware leicht verbreitet werden kann. Hierbei ist der Nutzer gefordert und er muss Entscheidungen treffen, die oft tiefgreifendes technisches Wissen voraussetzen. Damit er dieser Verantwortung gerecht werden kann, muss sichergestellt werden, dass er mit ausreichend Informationen über die Apps, die auf seinem System laufen, und deren Datennutzung versorgt wird. Nur so kann er die für ihn bestmöglichen Entscheidungen treffen. Auch muss das System über ein eingängiges Konfigurationssystem verfügen, das es dem Nutzer ermöglicht die getroffenen Entscheidungen entsprechend in die Tat umzusetzen.

Basierend auf den Erkenntnissen aus Abschnitt 2.4 kann jeder der drei Mobilplattformen iOS, Windows Phone und Android eines der drei Modelle zugeordnet werden [Sta13b]. In Abbildung 2.4 ist diese Einteilung dargestellt.

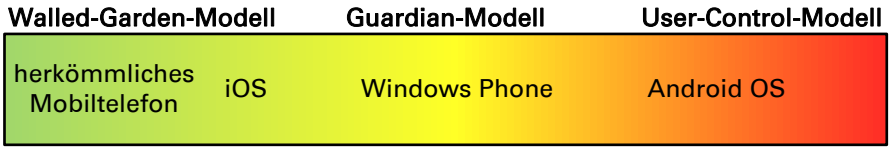


Abbildung 2.4: Klassifikation der heute vorherrschenden Mobilplattformen (in Anlehnung an [BV11; Sta13b])

Zur besseren Einordnung sind darin auch herkömmliche Mobiltelefone ohne die Möglichkeit weitere Apps zu installieren aufgeführt, da sie hinsichtlich der Gefährdung durch Schadsoftware als quasi sicher gelten.

iOS nimmt dem Nutzer nahezu alle relevanten Entscheidungen bezüglich der Verwendung von Apps ab und folgt damit dem Walled-Garden-Modell. Der Nutzer kann keine Apps verwenden, die von Apple nicht zugelassen sind, ohne Modifikationen am System vorzunehmen. Unter der Voraussetzung, dass die Überprüfung von Apps zuverlässig und im Sinne des Nutzers verläuft, bedarf es auf iOS keines weitreichenden Sicherheitssystems zum Schutz gegen Apps von Drittanbietern. Windows Phone bietet dem Nutzer mehr Freiheiten bezüglich der Verwendung von Apps. So lassen sich auf dieser Mobilplattform auch Apps ausführen, die von Microsoft nicht verifiziert wurden. Allerdings besitzen solche Apps nur stark eingeschränkte Berechtigungen. Ein solches Vorgehen entspricht weitestgehend dem Guardian-Modell. Auch wenn der Nutzer hier eine gewisse Entscheidungsfreiheit hat, limitiert Windows Phone das Gefahrenpotenzial dadurch, dass es selbstständig die Zugriffsrechte von kritischen Apps reduziert. Android folgt hingegen vollständig dem User-Control-Modell. Es bietet dem Nutzer die volle Kontrolle über das System, überträgt ihm damit aber auch sämtliche Verantwortung über die darauf befindlichen Daten. Dementsprechend benötigt Android ein anwenderfreundliches holistisches Sicherheitssystem, damit der Nutzer dieser Verantwortung gerecht werden kann.

Wie Shabtai et al. [SFK+10] in ihrer Studie allerdings feststellen, bietet das in Android implementierte Sicherheitssystem und vor allem das Berechtigungssystem diese Funktionen nicht: So können beispielsweise Apps installiert werden, obwohl der Nutzer dem nicht zustimmt. Auch ist der Nutzer häufig überfordert die Berechtigungen und deren Bedeutung zu verstehen und setzt

sich entsprechend gar nicht erst damit auseinander [FHE+12]. Dies ist allerdings dringend erforderlich, da viele Apps überprivilegiert sind [FCH+11] und sogar heimlich Berechtigungen untereinander austauschen [CHY12]. Dass es sich hierbei um keine Einzelfälle handelt, zeigen Enck et al. [EGC+10] und Enck et al. [EGH+14] in ihren Untersuchungen. So zeigten 20 von 30 zufällig ausgewählten Apps aus dem offiziellen Android-*App-Store Google Play* ein auffälliges Verhalten im Umgang mit sensiblen Daten. Unter auffälligem Verhalten sind u. a. die Weiterleitung von Nutzerdaten an Werbeserver, das Auslesen von identifizierenden Geräteinformationen wie der IMEI-Nummer oder die Sammlung von Daten über die Nutzung des Telefons subsumiert.

Aus diesem Grund bedarf insbesondere der Marktführer Android einer Überarbeitung und Erweiterung des Sicherheitssystems, um den speziellen Anforderungen gerecht zu werden, die ein User-Control-Modell mit sich bringt. Kapitel 3 untersucht daher speziell für Android, wie Sicherheitserweiterungen in das System eingebracht werden können, stellt beispielhaft unterschiedliche Ansätze zur Erweiterung des Sicherheitssystems von Android vor und vergleicht diese miteinander.

Kapitel 3

Analyse von Android-Datenschutzsystemen

Erweiterungen für die Datenschutzsysteme mobiler Plattformen sind aufgrund der in Kapitel 2 ermittelten Schwächen dieser Systeme regelmäßig Gegenstand der Forschung. Zur Realisierung dieser Erweiterungen werden diese entweder direkt in die mobile Plattform eingebracht (siehe Abschnitt 3.1) oder es werden alle Apps dahingehend manipuliert, dass diese mit den Erweiterungen interagieren können (siehe Abschnitt 3.2). Ein Vergleich dieser beiden Ansätze insbesondere hinsichtlich ihrer Eignung für die Realisierung des Datenschutzsystems *Privacy Management Platform (PMP)*, das im Rahmen der vorliegenden Arbeit in Kapitel 4 eingeführt wird, ist in Abschnitt 4.3.6 gegeben.

Neben diesen beiden Ansätzen, die überwiegend zur Realisierung eines Datenschutzsystems zum Einsatz kommen, gibt es noch weitere Ansätze, die in Abschnitt 3.3 besprochen werden.

Die im Folgenden gegebenen Beispiele und Architekturen basieren auf Android, da diese mobile Plattform nicht nur die größte Verbreitung sondern auch den größten Bedarf für eine Überarbeitung hat (siehe Abschnitt 2.4 und

Abschnitt 2.5). Die zugrundeliegenden Konzepte lassen sich jedoch auch auf andere mobile Plattformen anwenden, wie beispielsweise die Arbeiten von Agarwal und Hall [AH13], Bucicoiu et al. [BDDS15] und Werthmann et al. [WHD+13] belegen.

Dieses Kapitel basiert auf den eigenen Publikationen [SM13], [Sta13b] sowie [SM14].

3.1 Anpassung der mobilen Plattform

Ein naheliegender Ansatz, um ein neues Datenschutzsystem in eine mobile Plattform zu integrieren, besteht darin, dass die mobile Plattform selbst angepasst wird. Um weiterhin kompatibel zu bestehenden Apps zu bleiben, d. h. um sicherzustellen, dass diese Apps auch über das neue Datenschutzsystem Berechtigungen anfordern sowie Zugriff auf Nutzer- und Systemdaten erhalten können, muss das neue System nahtlos in den bestehenden Berechtigungsmechanismus eingebettet werden. Insbesondere bedeutet dies, dass sich an der Schnittstelle zu den Apps nichts ändern darf.

Scholz [Sch13b] analysiert im Rahmen seiner Diplomarbeit den Berechtigungsprozess von Android: Unter Android wird bei der Installation einer App vom Paketmanager die `scanPackageLI`-Methode aufgerufen. Von dieser Methode werden alle *Permissions*, die von der installierten App angefordert werden, ausgelesen. Dem Berechtigungsmodell von Android liegt eine Alles-oder-Nichts-Semantik zugrunde, d. h. wenn eine App vom Nutzer installiert wird, erteilt er ihr implizit alle angeforderten Berechtigungen. Daher stößt die `scanPackageLI`-Methode automatisch einen Prozess zur Persistierung aller ausgelesenen *Permissions* an. Dieser Prozess umfasst einerseits die Aktualisierung der im Speicher gehaltenen Berechtigungsregeln (über die `updateSettingsLI`-Methode) sowie die dauerhafte Speicherung der Berechtigungen als Extensible Markup Language (XML)-Dokument in einem geschützten Teil des Dateisystems (über die `writeLP`-Methode). Beim Systemstart werden in einem abgesicherten Prozess die Berechtigungen aus diesen Dateien ausgelesen und damit das Berechtigungsregelwerk im Speicher neu aufgebaut.

Der Installationsprozess kann von einem Datenschutzsystem dahingehend angepasst werden, dass der Nutzer einzelne *Permissions* deaktivieren und somit

```
1 private void enforce(
2         String permission, int resultOfCheck,
3         boolean selfToo, int uid, String message) {
4    if (resultOfCheck != PackageManager.PERMISSION_GRANTED) {
5       throw new SecurityException(...);
6    }
7 }
```

Quelltext 3.1: Implementierung der `enforce`-Methode in der `ContextImpl`-Klasse unter Android 5.1.1 (Auszug)

einer App bestimmte Berechtigungen entziehen kann. Neben einem veränderten Paketmanager-Installationsdialog, mit dem der Nutzer eine solche Auswahl treffen kann, ist es hierzu nötig die `scanPackageLI`-Methode anzupassen. Anstelle aller von einer App deklarierten *Permissions* an den nachgelagerten Persistierungsprozess weiterzugeben, darf diese Methode nur die vom Nutzer ausgewählte Untermenge weitergeben. Diese einfachen Anpassungen genügen bereits, um die Berechtigungen von Apps bei der Installation zu reglementieren.

Für Berechtigungsänderungen zur Laufzeit oder für die Einführung eines erweiterten Berechtigungsprozesses (z. B. durch die Einbeziehung von Kontextinformationen), sind jedoch weitreichendere Anpassungen nötig. Der Android-Berechtigungsprozess ist vereinfacht in Abbildung 3.1 dargestellt. Sobald eine App auf Methoden des Frameworks zugreift, wird vom Anwendungsframework diese Aufrufkette ausgelöst. Sind für den Aufruf der Methoden *Permissions* nötigt, wird aus dem Anwendungsframework heraus u. a. die `enforce`-Methode aufgerufen. Diese prüft, ob die App die benötigten *Permissions* besitzt und löst, falls dies nicht der Fall ist, eine Sicherheits-Exception aus. Die Implementierung dieser Methode ist auszugsweise in Quelltext 3.1 dargestellt.

Aufgrund Androids Alles-oder-Nichts-Semantik, wird davon ausgegangen, dass eine App alle Berechtigungen besitzt, die für die Ausführung dieser App nötig sind. Daher löst die `enforce`-Methode automatisch eine Sicherheits-Exception aus, wenn einer App gemäß des Berechtigungsregelwerks im Speicher eine benötigte *Permission* fehlt. Wird diese in der App nicht behandelt, führt dies zu einem Absturz der App.

Für die eigentliche Überprüfung, welche Berechtigungen eine App besitzt, wird die Anfrage an den `ActivityManager` weitergeleitet (über die `checkPermission`-Methode). Dieser ermittelt für die anfragende App u. a. den User Identifier (UID) sowie den Process Identifier (PID), über die eine App eindeutig identifiziert werden kann. Der `PackageManager` führt mit diesen Informationen die eigentliche Anfrage an das Berechtigungsregelwerk durch und gibt eine der beiden Integer-Konstanten `PERMISSION_DENIED` oder `PERMISSION_GRANTED` an die `enforce`-Methode zurück. Das Berechtigungsregelwerk ist als Positivliste aufgebaut, d. h. existiert für eine App-*Permission*-Kombination kein Eintrag, wird die Berechtigungsanfrage zurückgewiesen.

Für die Integration eines neuen Datenschutzsystems stellt jede der drei Klassen (`Context`, `ActivityManager` und `PackageManager`) einen potentiellen Einstiegspunkt dar. In der `enforce`-Methode kann ein neues System dafür sorgen, dass für den Fall, dass einer App eine nötige Berechtigung fehlt, in dieser keine Sicherheits-Exception ausgelöst wird. Stattdessen kann beispielsweise der Nutzer ein Feedback erhalten, dass für die Ausführung der App, bzw. eines bestimmten Features der App, weitere Berechtigungen benötigt werden. Die `enforce`-Methode kann dem Nutzer auch exakt angeben, um welche *Permissions* es sich dabei handelt. In diesen Feedback-Dialog kann eine Komponente integriert sein, über die der Nutzer die Berechtigungen der App rekonfigurieren kann. Diese Komponente müsste im Anschluss erneut die `updateSettingsLI`-Methode und `writeLP`-Methode aufrufen, damit die Änderungen übernommen werden. Im `ActivityManager` kann das Datenschutzsystem weitere Informationen zu der anfragenden App abrufen oder zu dem Kontext, in dem die App eine Berechtigung anfordert. Auf diese Weise kann das neue Datenschutzsystem beispielsweise kontextsensitive Berechtigungsregeln einführen. Der `PackageManager` kann manipuliert werden, wenn tiefgehende Anpassungen am Berechtigungssystem durchgeführt werden sollen. Beispielsweise können dadurch neben den beiden Konstanten `PERMISSION_DENIED` und `PERMISSION_GRANTED` weitere Werte definiert werden, um eine Berechtigung nur unter gewissen Auflagen zu genehmigen (z. B. wie oft eine App eine Methode ausführen darf oder wie lange eine Berechtigung gültig ist).

Beispiele für Datenschutzsysteme, die diesem Ansatz folgen sind u. a. *Apex* von Nauman, Khan und Zhang [NKZ10], *AppFence* von Hornyack et al.

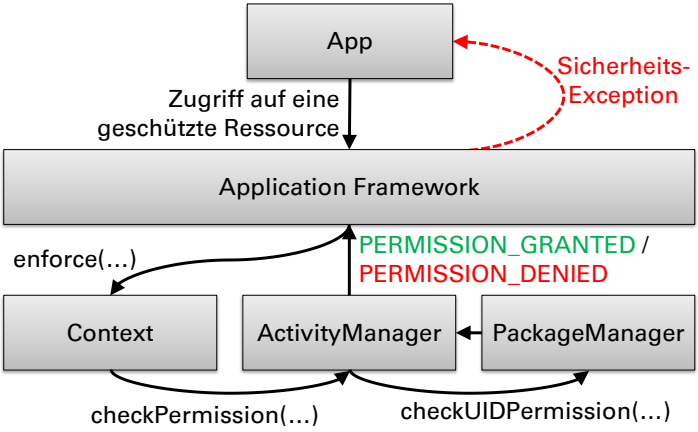


Abbildung 3.1: Verarbeitung einer Berechtigungsanfrage unter Android (in Anlehnung an [Sta15a])

[HHJ+11], *CRêPE* von Conti, Nguyen und Crispo [CNC11] und Conti et al. [CCFZ12], *Data-Sluice* von Saracino et al. [SMAD16], *IacDroid* von Zhang et al. [ZWL+16], *MockDroid* von Beresford et al. [BRSS11], *SEAF* von Banuri et al. [BAK+12], *Sorbet* von Fragkaki et al. [FBJS12] sowie *YAASE* von Russello et al. [RCFZ11]. Diese Systeme werden in Abschnitt 4.5 genauer betrachtet und miteinander verglichen.

Ein solcher Ansatz ist nicht nur intuitiv, sondern er bietet auch den Vorteil, dass durch die tiefe Integration in die mobile Plattform das neue Datenschutzsystem die gleichen Rechte und Möglichkeiten besitzt, als wäre es ein orginärer Bestandteil der Plattform. Außerdem findet das neue Datenschutzsystem unmittelbar Anwendung auf alle Apps, die auf der Plattform installiert werden. Das Hauptproblem, das sich aus der Manipulation der mobilen Plattform ergibt, ist jedoch, dass der Nutzer die vollständige Plattform bei sich neu aufsetzen muss, wenn er das neue Datenschutzsystem nutzen möchte. Einerseits bedeutet dies, dass das Datenschutzsystem für jede mögliche Kombination aus Hardware und Android-Version verfügbar sein muss[1]. Andererseits werden viele Nutzer abgeschreckt, da der dadurch entstehende Installationsaufwand sehr

[1] Erschwerend kommt hinzu, dass viele Smart-Device-Hersteller individuelle Anpassungen an Android durchführen, wodurch die Anzahl an Kombinationsmöglichkeiten weiter steigt.

hoch ist (u. a. werden *Root*-Rechte benötigt und geschützte Speicherbereiche müssen überschrieben werden), das Datenschutzsystem selbst Schadsoftware enthalten kann und der Garantieanspruch durch die Manipulation verloren geht.

Backes et al. [BBGvS14] führen daher einen Ansatz ein, der diesen Problemen entgegenwirkt. Das *Android Security Framework* (*ASF*) ist ein generisches Framework, das einmal in die Android-Systemarchitektur in die Nahtstelle zwischen dem Anwendungsframework und den Android-Diensten (siehe Abbildung 2.3) eingebracht werden muss und anschließend die Realisierung von beliebigen Datenschutzsystemen ermöglicht, ohne dass dafür weitere Manipulationen an der mobilen Plattform nötig sind. Angelehnt an die Arbeiten von Wright et al. [WCS+02] und Watson et al. [WMVF03], die vergleichbare Sicherheitsframeworks für Linux-basierte Systeme vorstellen, verfolgt *ASF* zwei inhärente Leitgedanken. (1) Die Entwickler des Datenschutzsystems müssen zur Definition ihres Regelwerks auf eine Turing-vollständige formale Sprache zurückgreifen können. Dadurch sind sie in der Lage ein eigenes Berechtigungsmodell sowie Regeln, wie dieses ausgewertet wird, in der Sprache anzugeben. Somit sind sie vollständig unabhängig von dem Modell der zugrundeliegenden mobilen Plattform und gegen Änderungen dieses Modells (z. B. bei einem Versionswechsel der Plattform) immun. (2) Das Sicherheitsframework muss vollständig unabhängig von dem Regelwerk und dem Berechtigungsmodell des Datenschutzsystems auf der einen sowie dem der mobilen Plattform auf der anderen Seite sein. Insbesondere bedeutet dies, dass die Logik und Umsetzung eines neuen Datenschutzsystems sich ausschließlich in Modulen befindet, die außerhalb des Sicherheitsframework liegen.

Das *ASF* sorgt für die Durchsetzung der von den Datenschutzsystementwicklern definierten Regeln sowie für die dafür nötige Kommunikation mit der mobilen Plattform. Da das *ASF* den beiden obigen Leitgedanken vollständig entspricht, sind die Entwickler von Datenschutzsystemen bei der Realisierung ihres Systems nicht eingeschränkt. Mit dem *ASF* können sie ihr Datenschutzsystem als Erweiterung für eine mobile Plattform realisieren, ohne deren Komponenten manipulieren zu müssen. Die Nutzer profitieren hiervon unmittelbar, da sie beliebige Datenschutzsysteme gefahrlos ausprobieren können, ohne bei jedem Wechsel die mobile Plattform von Grund auf neu aufsetzen zu müssen. Ein

System wie *ASF* kann zur Realisierung der im Rahmen der vorliegenden Arbeit entwickelten Konzepte genutzt werden.

3.2 Anpassung der Apps

Während die Anpassung der mobilen Plattform zur Integration eines neuen Datenschutzsystems in der Theorie sehr effektiv und naheliegend erscheint, stellt in der Praxis der daraus resultierende komplexe Installationsprozess für viele Nutzer ein Ausschlusskriterium dar [STCT15]. Außerdem zeigen Untersuchungen, dass nach der Manipulation einer mobilen Plattform diese häufig mehr Sicherheitsprobleme aufzeigt als zuvor [WGZ+13; GHD+15].

Aus diesem Grund wählen viele Entwickler von Datenschutzsystemen einen anderen Weg, um ihr System in eine bestehende mobile Plattform zu integrieren. Anstelle die mobile Plattform respektive deren Berechtigungssystem zu modifizieren, wird das neue Datenschutzsystem als App eingeführt. Dieser Ansatz ist vergleichbar mit der Einführung eines Virenscanners für Desktop PCs. Da allerdings auf mobilen Plattformen jede App in einer strikt isolierten Sandbox abläuft und daher nur bedingt Einfluss auf andere Apps nehmen kann, ist zusätzlich eine Anpassung am Programmcode jeder App erforderlich. Das Datenschutzsystem bietet Schnittstellen an, über die Berechtigungsanfragen oder Datenzugriffe erfolgen können. Apps von Drittanbietern müssen ihren Berechtigungsprozess dahingehend anpassen, dass sie ihre Berechtigungen oder Daten nicht mehr über das Anwendungsframework der mobilen Plattform anfordern, sondern von dem neuen Datenschutzsystem.

Diese Anpassung kann seitens des Entwicklers auf Ebene des Quellcodes erfolgen. Durch die Einführung der *Runtime Permissions* in Android wurde es beispielsweise erforderlich, dass der Entwickler vor Methodenaufrufen, die eine spezielle Berechtigung benötigen, zunächst prüft, ob die App aktuell die zugehörige *Permission* besitzt. Der hierfür nötige Code ist in Quelltext 3.2 dargestellt.

Zunächst muss die App beim Paketmanager anfragen, ob sie die benötigte *Permission* besitzt und falls nicht muss sie diese vom Nutzer explizit anfordern. Um die Entscheidung des Nutzers behandeln zu können, muss zusätzlich die

```
1 ...
2 if (ContextCompat.checkSelfPermission(thisActivity,
   ↪  Manifest.permission.BLUETOOTH) !=
   ↪  PackageManager.PERMISSION_GRANTED) {
3    ActivityCompat.requestPermissions(thisActivity, new
      ↪  String[]{Manifest.permission.BLUETOOTH},
      ↪  MY_BT_REQUEST);
4 }
5 ...
```

Quelltext 3.2: *Permission*-Prüfung zur Laufzeit in Android 6.0

`onRequestPermissionsResult`-Methode überschrieben werden. Darin muss auf Rückmeldungen zu dem Schlüssel `MY_BT_REQUEST` reagiert werden.

Dies erzeugt einen sehr großen Overhead für die Entwickler. Nicht nur die Menge des zu schreibenden Codes steigt stark an, sondern es wird zusätzlich Wissen darüber benötigt, welche *Permissions* welchen Methodenaufrufen zugeordnet sind. Während Google als Anbieter der mobilen Plattform durchsetzen kann, dass Entwickler sich an diese Codeanpassungen halten, haben die Anbieter eines Datenschutzsystems keine solche Machtstellung. Daher fokussieren sich diese verstärkt darauf, welche Anpassungen auf Ebene des Bytecodes einer App möglich sind. Solche Anpassungen können ohne die Mithilfe des App-Entwicklers durch den Nutzer selbst durchgeführt werden[2]. Abbildung 3.2 skizziert am Beispiel einer Android-App, welche Anpassungen durchgeführt werden können.

Wie in der Abbildung zu sehen ist, gibt es primär vier Komponenten, die für die Manipulation in Frage kommen, wenn es darum geht, ein neues Datenschutzsystem einzubinden. Die wesentlichsten Anpassungen müssen im Dalvik-Bytecode (*classes*) einer App durchgeführt werden. In diesem befinden sich die Methodenaufrufe an das Anwendungsframework, die durch das neue Datenschutzsystem reglementiert werden sollen. Grundsätzlich gibt es zwei unterschiedliche Strategien, wie dies realisiert wird. Entweder werden sämtliche kritische Methodenaufrufe durch äquivalente Aufrufe an das Datenschutzsys-

[2]Eine Werkzeugunterstützung seitens des Anbieters des Datenschutzsystems wird hierbei vorausgesetzt.

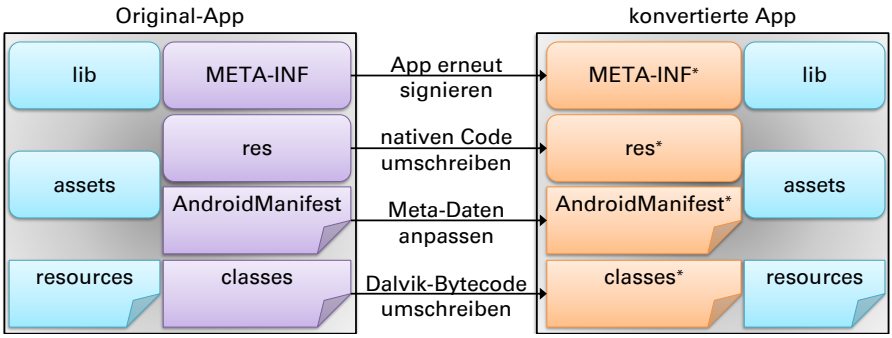


Abbildung 3.2: Arbeitsprinzip eines App-Konverters (in Anlehnung an [Sta13b])

tem ersetzt, das anschließend die Berechtigungsprüfung sowie den Datenzugriff regelt, oder es werden *Inline-Reference-Monitoring*-Komponenten [Erl04] in die App eingebaut. Mit diesen Monitoring-Komponenten überwacht eine App sich selbst und fordert bei Bedarf Berechtigungen vom Datenschutzsystem an. Der eigentliche Programmcode bleibt allerdings unverändert, d. h. die Methodenaufrufe richten sich weiterhin an das Anwendungsframework; diese werden jedoch ggf. von der Monitoring-Komponenten abgeblockt oder mit falschen Rückgabewerten versorgt. Neben dem Dalvik-Bytecode können auch die nativen Codefragmente (*res*) einer App adaptiert werden. Hierzu zählen alle Aufrufe, die sich direkt an die hardwarenahen Schichten der Android-Systemarchitektur richten.

Neben diesen beiden Komponenten, die die Programmlogik beinhalten, muss bei der Anpassung einer App an ein neues Datenschutzsystem auch das Manifest der App manipuliert werden (*AndroidManifest*). So können selbstdefinierte *Permissions* eingetragen und zusätzliche Aktivitäten oder Services deklariert werden, wenn dies für das neue Datenschutzsystem oder dessen Monitoring-Komponenten nötig ist. Da durch Manipulationen auf Ebene des Bytecodes die Signatur einer App ungültig wird, muss diese anschließend neu signiert werden (*META-INF*).

Davis und Chen [DC13] ermittelten jedoch zwei wesentliche Probleme, die sich bei diesem Prozess in der Praxis ergeben:

Der Android-Verifizierer. Bei der Kompilierung einer App werden Codeoptimierungen und Variablenindexierungen durchgeführt. Der Android-Verifizierer stellt zur Laufzeit sicher, dass bei diesen Indizes keine Duplikate auftreten. Wird jedoch vorkompilierter Code in den Bytecode einer bestehenden App eingefügt, so kann es zu Anomalien an dieser Indexstruktur kommen. Dadurch wird die Ausführung der App durch den Android-Verifizierer verhindert.

Die Einbindung von Bibliotheken. Werden von dem eingefügten Code neue Bibliotheken angesprochen, so kann dies zu einem Problem führen. Unter Android können externe Bibliotheken erst angesprochen werden, wenn die jeweilige App vollständig initialisiert wurde, d. h. nach Abarbeitung der `onCreate`-Methode. Diese Methode stellt den Startpunkt einer App dar. Das neue Datenschutzsystem kann innerhalb dieser Methode keine Aufrufe überwachen, wenn hierfür externe Bibliotheken nötig sind.

Reddy et al. [RJV+11] beschreiben, wie trotz dieser Einschränkungen die Manipulation von Dalvik-Bytecode möglich ist. Sheehan, Davis und Chen [SDC13] demonstrieren die Funktionsweise eines Werkzeugs zur Manipulation von Android-Apps. In Abschnitt 4.3.5 ist der Aufbau einer Werkzeugkette gegeben, mit der alle obigen Manipulationsschritte durchgeführt werden können.

Beispiele für Datenschutzsysteme, die diesem Ansatz folgen sind u. a. *AppGuard* von Backes et al. [BGH+12], *AUDACIOUS* von Ringer, Grossman und Roesner [RGR16], *Aurasium* von Xu, Saıdi und Anderson [XSA12], *Dr. Android & Mr. Hide* von Jeon et al. [JMV+12], *I-ARM Droid* von Davis et al. [DSKC12] und *RetroSkeleton* von Davis und Chen [DC13]. Diese Systeme werden in Abschnitt 4.5 genauer betrachtet und miteinander verglichen.

Der Vorteil bei diesem Ansatz liegt klar auf der Hand, da der Nutzer hierfür keine Modifikationen an der mobilen Plattform vornehmen muss und das Datenschutzsystem einfach wie eine App installieren kann. Allerdings zeigt eine Evaluation von Hao, Singh und Du [HSD13], dass ein Großteil dieser Systeme an Kinderkrankheiten leidet. So müssen beispielsweise alle Apps zunächst auf das neue Datenschutzsystem angepasst werden. Diese Aufgabe bleibt oft dem Nutzer überlassen, der sich dessen nicht unbedingt bewusst ist und sich somit in falscher Sicherheit wiegt. Vergisst er eine App anzupassen, so verwendet diese weiterhin das alte, unsichere Datenschutzsystem. Darüber hinaus erhalten Apps nach der Manipulation keine automatischen Updates

mehr, wodurch Sicherheitslücken nicht geschlossen werden [EOM09]. Auch bei der eigentlichen Manipulation des Bytecodes können Fehler entstehen, die zu neuen Sicherheitsproblemen führen.

Neben diesen eher technischen Problemen, besitzt dieser Ansatz ein inhärentes rechtliches Problem [APW17]. Die Manipulation einer App verstößt gegen das Urheberrecht des Entwicklers gemäß § 69a (1) – (3) UrhG (Gegenstand des Schutzes) [BRD16a]. Auch wenn kleinere Manipulationen für den privaten Gebrauch teilweise als zulässig angesehen sind, würde eine automatische und dauerhafte Konvertierung einer App zusätzlich gegen § 14 UrhG (Entstellung des Werkes) [BRD16a] sowie § 69c (2) UrhG (Zustimmungsbedürftige Handlungen) [BRD16a] verstoßen. Darüber hinaus ist sowohl die Zurverfügungstellung als auch die Nutzung eines solchen Werkzeugs strafrechtlich bedenklich. § 202a StGB (Ausspähen von Daten) [BRD16b] untersagt den Zugriff auf geschützte Daten[3] und gemäß § 202c StGB (Vorbereiten des Ausspähens und Abfangens von Daten) [BRD16b] ist auch die Erstellung eines Programms, mit dem ein solcher Zugriff möglich ist, eine Straftat. § 303a StGB (Datenveränderung) [BRD16b] legt fest, dass neben der eigentlichen Manipulation von Daten bereits der Versuch oder die dafür nötige Vorbereitung strafbar ist. Aus diesem Grund werden im Rahmen der vorliegenden Arbeit nur die theoretischen Grundlagen eines solchen Ansatzes behandelt.

3.3 Weitere Ansätze

Neben diesen beiden Ansätzen zur Realisierung eines Datenschutzsystems, gibt es noch weitere Herangehensweisen, um den Datenschutz auf mobilen Plattformen zu verbessern. Im Folgenden werden die im Kontext der vorliegenden Arbeit relevanten Ansätze besprochen. Dabei werden nur ausgewählte, repräsentative Vertreter behandelt, da eine vollständige Übersicht in Anbetracht der großen Menge nicht gegeben werden kann. Diese Ansätze stellen keine Alternativen zu den in Abschnitt 3.1 und Abschnitt 3.2 vorgestellten Datenschutzsystem dar, sondern können ergänzend neben einem solchen System genutzt werden.

[3] Der Bytecode einer App gilt durch die Signatur als geschützt.

Motiee, Hawkey und Beznosov [MHB10] untersuchen am Beispiel von Windows Vista, wie Nutzer auf Berechtigungsanfragen reagieren. Die Untersuchungen zeigen, dass Nutzer Anwendungen nahezu immer Administratorrechte geben, ohne zu hinterfragen, wozu sie diese Rechte benötigen. Da Windows Vista genau wie viele mobile Plattformen das *Least-Privilege*-Prinzip umsetzt, lassen sich die Erkenntnisse auf diese Plattformen übertragen. Felt et al. [FHE+12] bestätigen dies und zeigen, dass sich Nutzer den potentiellen Gefahren, die von einigen Berechtigungen ausgehen können, nicht bewusst sind. Sarma et al. [SLG+12] führen daher eine Beschreibungssprache für Android-*Permissions* ein. Mittels dieser Sprache lässt sich spezifizieren, wozu eine App, die eine bestimmte *Permission* besitzt, in der Lage ist. Hierbei lassen sich auch Wechselwirkungen zwischen mehreren *Permissions* angeben sowie besonders kritische Kombinationen beschreiben. Die Beschreibungen sollen Nutzern helfen zu verstehen, wozu sie eine App berechtigen. Außerdem lassen sich so Vergleiche ziehen, ob verwandte Apps bei einem ähnlichen Funktionsumfang weniger Berechtigungen benötigen.

Während es sich hierbei um eine statische Analyse basierend auf den *Permissions* einer App handelt, untersuchen Enck et al. [EGC+10] den Informationsfluss einer App. Mit ihrem Analysewerkzeug *TaintDroid* kann zur Laufzeit einer App überprüft werden, wie sensible Daten von ihr verarbeitet werden. Diese Analyseergebnisse dienen als Eingabedaten für ein Schutzsystem, mit dem die Verwendung von Daten sowie der Datenaustausch zwischen Apps reglementiert werden kann, wie es von Kodeswaran et al. [KNK+12] vorgestellt wird.

Neben Daten können Apps allerdings auch Berechtigungen austauschen. Mit *DroidAlarm* [ZXMX13], *DroidChecker* [CHY12] und *WeChecker* [CWH+15] werden Entwicklern Werkzeuge an die Hand gegeben, mit denen diese überprüfen können, ob eine App Berechtigungen weitergeben kann. Diese Werkzeuge untersuchen zunächst statisch, welche Berechtigungen eine App besitzt und anschließend dynamisch, ob diese anderen Apps über nicht abgesicherte Schnittstellen unbewusst zur Verfügung stehen. Im Gegensatz zu diesen Arbeiten, schlagen Gilbert et al. [GCCJ11] mit *AppInspector* ein System vor, das durch statische Analysen des Manifests und dynamische Analysen zur Laufzeit potentielle Angreifer ermittelt. Dieses Werkzeug richtet sich an Nutzer, die dadurch vor der Installation von Schadsoftware gewarnt werden.

Dies ist auch das Ziel von Spreitzenbarth et al. [SFE+13]. Mit *Mobile Sandbox* führen sie ein System ein, das Schadsoftware ebenfalls mithilfe von einer Kombination aus statischen und dynamischen Analysen ermitteln soll. Dieses System geht jedoch wesentlich weiter, da es in der statischen Analysephase zunächst die App dekompiliert und auf dem so teilweise wiederhergestellten Quellcode nach verdächtigem Code sucht. In der dynamischen Analyse wird die App in einem Emulator ausgeführt. Dieser zeichnet alle Operationen auf, die von der App durchgeführt werden. Dadurch erfasst *Mobile Sandbox*, im Gegensatz zu den obigen Systemen, sogar die Bestandteile der App, die in nativem Code geschrieben wurden. *Paranoid Android* von Portokalidis et al. [PHAB10] wählt einen ähnlichen Weg, jedoch wird dabei die Überprüfung auf einem externen Server ausgeführt, auf dem ein Android-Emulator mit Analysefunktionen läuft. Auf diese Weise muss ein Nutzer eine potentiell schädliche App zur Überprüfung nicht einmal auf sein physisches Smart Device laden.

Einen anderer Ansatz zur Bewertung der Vertrauenswürdigkeit einer App wählen Jin et al. [JSCK13]. In ihrem System *Privacy Agent* kommen hierzu Crowdsourcing-Techniken zum Einsatz. Auf einem externen Server werden Empfehlungen von Nutzern gesammelt, welche Berechtigungen ein bestimmter Typ von App besitzen sollte. Zusätzlich analysiert *Privacy Agent* den Umgang eines Nutzers mit Berechtigungen und hinterlegt diesen lokal auf dem Smart Device. Basierend auf diesen persönlichen Vorlieben und den Empfehlungen aus der Crowd bestimmt *Privacy Agent* für jede neue App, welche Berechtigungseinstellungen für den jeweiligen Nutzer angebracht sind. Diese Einstellungsempfehlung kann entweder als Entscheidungshilfe für den Nutzer dienen, ob dieser die App installieren sollte, oder als Eingabedaten für ein nachgelagertes Datenschutzsystem, mit dem einer App Berechtigungen auch feingranular entzogen werden können.

Um die Ausführung von Apps einschränken zu können, führen Becker, Fournet und Gordon [BFG10] mit *SecPAL* eine Sprache zur Beschreibung von Berechtigungen ein. Mit *SecPAL* können Berechtigungen mit Kontextrestriktionen verknüpft werden. Das Android-Werkzeug *AppPAL* von Hallett und Aspinall [HA16] verwendet diese Sprache, um ein Regelwerk aufzustellen, welche Apps in welcher Situation ausgeführt werden dürfen. Mittels *SecPAL* kann der Nutzer hierzu zunächst bestimmte Situationen beschreiben (beispielsweise „Arbeit“

oder „Freizeit“). In *AppPAL* können auch Apps entsprechend ihrer Eingenschaften, insbesondere deren *Permissions* gruppiert werden. So könnte eine Regel lauten, dass im Kontext „Arbeit“ keine App ausgeführt werden darf, die die *Permission* `SEND_SMS` besitzt. Die Berechtigungen einer App lassen sich mit *AppPAL* allerdings nicht reglementieren.

England, DeTreville und Lampson [EDL01] beschreiben, wie man Digital Rights Management (DRM)-Techniken dafür einsetzen kann, den Zugriff, den eine App auf Daten bekommt, feingranular zu beschränken. So lassen sich für jedes Datum unterschiedliche Zugriffsarten definieren (u. a. Lesen, Schreiben, Löschen, ...) und für jede Zugriffsart weitere Restriktionen angeben (z. B. wie oft eine App diesen Zugriff auf das Datum ausführen darf). Das Alleinstellungsmerkmal dieses Ansatzes besteht allerdings darin, dass die Zugriffsrechte an das jeweilige Datum gebunden sind. Wenn eine App *A* ein Datum einliest und es einer App *B* zur Verfügung stellt, so besitzt *B* auf dem Datum nur die Schnittmenge der Berechtigungen von *A* und *B*. England, DeTreville und Lampson stellen allerdings nur die Technik und keine Implementierung davon vor. Für eine solche wären zunächst weitreichende Veränderungen an dem Datenmodell der mobilen Plattform nötig, was der vollständigen Überarbeitung der mobilen Plattform gleichkommt.

Da von Android sehr viele Derivate existieren, stellen Zheng, Sun und Lui [ZSL14] mit *DroidRay* ein Analysewerkzeug zur Verfügung, mit dem sich die Sicherheit eines Android-Systems überprüfen lässt. Die Gefahren stammen bei einem solchen System aus zweierlei Quellen. Einerseits kann es bei den Anpassungen an der mobilen Plattform selbst zu Fehlern kommen, wodurch neue Sicherheitslücken entstehen, und wichtige Sicherheits-Updates werden für die Derivate nicht oder nur stark verspätet umgesetzt. Andererseits sind in jedem Derivat mehrere Apps im Lieferumfang inbegriffen, die sehr viel mehr Rechte besitzen, als Apps, die vom Nutzer nachträglich installiert werden. Wird daher eine Schadsoftware mit einem Derivat ausgeliefert, so ist der potentielle Schaden, der entstehen kann, erheblich größer. Mit *DroidRay* kann beides untersucht und bewertet werden.

Mit *Boxify* stellen Backes et al. [BBH+15] einen Entwurfsansatz für Datenschutzsysteme vor, der weder Anpassungen an einer mobilen Plattform erforderlich macht, noch Werkzeuge zur Manipulation von Apps benötigt. In

diesem Ansatz wird ein Android-Feature namens *Isolated Process* ausgenutzt. Eine App kann einige ihrer Services in einem *Isolated Process* ausführen, die dadurch selbst keine Berechtigungen mehr besitzen und vollständig vom restlichen System abgeschottet sind [And16a]. Die *Boxify*-Komponente *Target* wird in einem solchen Prozess ausgeführt. Sie implementiert einen Sandbox-Dienst, in dem beliebige Apps ausgeführt werden können. Durch die Isolation von *Target* ist sichergestellt, dass eine App, die in der Sandbox läuft, ebenfalls keine Berechtigungen besitzt. Um einer App feingranular Berechtigungen zu erteilen, kommt eine zweite Komponente von *Boxify* zum Einsatz, der *Broker*. Dieser ist eine App, die alle verfügbaren Berechtigungen besitzt. Dadurch ist er in der Lage, andere Apps an das *Target* zu binden und diesem das Lifecycle-Management zu übertragen. Ab diesem Punkt fungiert das *Target* als *Shim*[4]. Sämtliche Aufrufe der App (sowohl Aufrufe an das Anwendungsframework als auch native Aufrufe) werden vom *Target* abgefangen und an den *Broker* weitergeleitet. Der *Broker* prüft anschließend, wie er den jeweiligen Aufruf zu verarbeiten hat und leitet ggf. das Ergebnis zurück an das *Target*. In *Boxify* ist die Logik des *Brokers* nicht vorgegeben, d. h. es handelt sich dabei zunächst um kein Datenschutzsystem. Allerdings wäre es möglich, ein solches im *Broker* zu realisieren.

Da keines der untersuchten Systeme die Anforderungen an ein Datenschutzsystem vollständig erfüllt, wird in Kapitel 4, basierend auf den Erkenntnissen zum Stand der Forschung, mit der *PMP* der Entwurf eines feingranularen und kontextsensitiven Datenschutzsystems vorgestellt, mit dem der Nutzer zur Laufzeit einer App dieser Berechtigungen entziehen oder gewähren kann. Zusätzlich ist er durch die *PMP* in der Lage, seine Daten zu anonymisieren oder nur aggregiert bzw. gefiltert an eine App weiterzugeben. Damit er in der Lage ist, die für ihn optimale Konfiguration der Regeln zu finden, versorgt die *PMP* ihn stets mit ausreichend Feedback zu seinen Einstellungen. Bei der Wahl der besten Implementierungsstrategie werden die Erkenntnisse aus Abschnitt 3.1 und Abschnitt 3.2 berücksichtigt.

[4]Unter einem *Shim* versteht man eine Softwarekomponete, die Application Programming Interface (API)-Aufrufe einer Anwendung an das darunterliegende Betriebssystem anpasst, um die Kompatibilität der Anwendung zu erhöhen (siehe Newton [New11]).

Kapitel 4

Die Privacy Management Platform

Rundum erfüllt keines der in den vorherrschenden mobilen Plattformen implementierten Sicherheitsmodelle die in Abschnitt 1.3 definierten Schutzziele, wie die Analyse in Kapitel 2 zeigt. Auch die in Kapitel 3 analysierten Android-Datenschutzsysteme von Drittanbietern erfüllen diese Schutzziele nicht. Daher wird im Folgenden mit der *Privacy Management Platform (PMP)* ein alternatives holistischen Sicherheitssystem eingeführt. Obwohl die Analyse zeigt, dass Android bezüglich Datensicherheit und Datenschutz sowohl iOS als auch Windows Phone unterlegen ist, überzeugt das von Google verfolgte Grundziel, dem Nutzer die Kontrolle über seine Daten zu geben und ihm sämtliche Freiheiten zu lassen. Außerdem besitzt Android den höchsten Verbreitungsgrad, weshalb es besonders geschützt werden muss. Daher sollen die im Rahmen dieser Arbeit entwickelten Konzepte die Offenheit von Android beibehalten und keine spezielle Hardware oder proprietäre Software voraussetzen [Lud13]. Zusätzlich soll dabei das *Least-Privilege*-Prinzip berücksichtigt werden, um den potentiellen Schaden, den eine bösartige App verursachen kann, zu minimieren. Die der

PMP zugrundeliegenden Konzepte und Modelle lassen sich jedoch auch auf andere Plattformen anwenden, die mit vielen privaten Daten arbeiten.

Die vorliegende Arbeit befasst sich ausdrücklich mit dem Schutz von Daten auf dem Gerät gegen illegale oder unerwünschte Zugriffe von Apps. Einen Schutz über die Gerätegrenze hinweg (z. B. analog zu den Datennutzungsreglementierungen von Digital Rights Management (DRM), siehe Rosenblatt, Mooney und Trippe [RMT01]) ist dabei explizit nicht vorgesehen. Für Lösungen zum Schutz von Daten, nachdem sie das Gerät verlassen haben, sei auf die Arbeiten von beispielsweise Neisse, Pretschner und Di Giacomo [NPD13], Kumari und Pretschner [KP13b], Kelbert und Pretschner [KP14], Kelbert und Pretschner [KP13a], Lovat und Pretschner [LP11] oder Den Hartog und Zannone [dHZ16] verwiesen.

Initial wird in Abschnitt 4.1 zunächst mit dem *Privacy Policy Model (PPM)* ein vollständig neues Modell zur Formulierung von Berechtigungsregeln eingeführt. Sowohl die in Windows Phone verwendeten *Capabilities* als auch die in Android zum Einsatz kommenden *Permissions* sind grobgranular und statisch. Entwickler können ihre Apps dahern nicht nach dem *Least-Privilege*-Prinzip erstellen. Für Nutzer stellen sie keinen wirksamen Mechanismus dar, mit dem diese sich um den Datenschutz (gemäß der Definition in Abschnitt 1.2.2) kümmern können. Anschließend wird in Abschnitt 4.2 auf die sieben elementaren Eigenschaften der *PMP* eingegangen, bevor Abschnitt 4.3 mögliche Implementierungsstrategien für die *PMP* vorstellt. Durch die Veränderungen am Modell der Berechtigungsregeln, werden auch Anpassungen am App-Modell nötig, damit die Apps flexibel auf Regeländerungen durch den Nutzer reagieren können. Damit die *PMP* abwärtskompatibel zu Apps bleibt, die nicht diesem neuen App-Modell folgen, und dem Nutzer dennoch einen verbesserten Schutz seiner Daten bieten kann, wurde mit dem *PMP-Gatekeeper* eine Komponente geschaffen, die diese Kompatibilität sicherstellt (siehe Abschnitt 4.4). Abschließend werden in Abschnitt 4.5 verwandte Arbeiten vorgestellt, die ähnlich wie die *PMP* alternative und erweiterte Berechtigungsmodelle und -systeme für mobile Plattformen vorstellen.

Das *PPM* wurde in den eigenen Publikationen [SM13] sowie [Sta13b] eingeführt. Die Eigenschaften der *PMP* wurden mittels der eigenen Publikationen [Sta13a] demonstriert. Die Implementierungsstrategien wurden in

den eigenen Publikationen [SM14] sowie [Sta13b] diskutiert, und der *PMP-Gatekeeper* wurde in der eigenen Publikation [Sta15a] vorgestellt. Eine vollständig auf Android 6.0 angepasste Version der *PMP* wurde in [Sta15b] veröffentlicht.

Zu der Implementierung des *PPMs* sowie des ersten Prototyps der *PMP* trug das Studienprojekt von Vetter et al. [VJB+12] maßgeblich bei. Die Integration der *PMP* in Android wurde zum Teil im Rahmen der Diplomarbeit von Scholz [Sch13b] durchgeführt und der *PMP-Gatekeeper* wurde im Rahmen der Bachelorarbeit von Salsa [Sal14] implementiert.

4.1 Das Privacy Policy Model

Um dem *Least-Privilege*-Prinzip gerecht zu werden, muss ein Berechtigungsmodell sowohl feingranular als auch flexibel (im Sinne von anpassbar auf den Kontext) sein. Die *Capabilities* in Windows Phone und die *Permissions* in Android bündeln häufig mehrere Berechtigungen. Z. B. erlaubt die *Permission* `Bluetooth` einer App diverse Funktionen mit einem per Bluetooth verbundenen Gerät auszuführen, ohne dass der Nutzer weiß, welche Funktionen genau benötigt wird oder gar mit welchem Gerät oder zumindest Gerätetyp diese von der App ausgeführt werden. Das *PPM* verfolgt daher einen komplett anderen Weg. Sowohl die Apps als auch die zu schützenden Einheiten werden logisch in kleinere Funktionseinheiten aufgeteilt. Dadurch werden im *PPM* nicht Apps Zugriffsrechte auf Daten oder Systemfunktionen eingeräumt, sondern es werden jeweils einzelne Funktionseinheiten miteinander verknüpft. So kann beispielsweise für einen Gerätetyp, der per Bluetooth mit einem Smartphone verbunden ist, eine konkrete Funktion mit Zugriffsrechten versehen werden. Dadurch kann der Nutzer sehr feingranular den Leistungsumfang einer App bedarfsgerecht erweitern oder reduzieren und somit gleichzeitig deren Zugriffsrechte ausweiten oder einschränken.

Wie die einzelnen Komponenten des *PPMs* miteinander in Beziehung stehen ist in Abbildung 4.1 in einer Unified Modeling Language (UML)-artigen Notation dargestellt. Auf die Komponenten wird in den folgenden Abschnitten näher eingegangen[1]. Wie in der Abbildung zu sehen ist, geht das *PPM* und damit

[1] Die *Presets* werden in Abschnitt 4.2.7 gesondert besprochen.

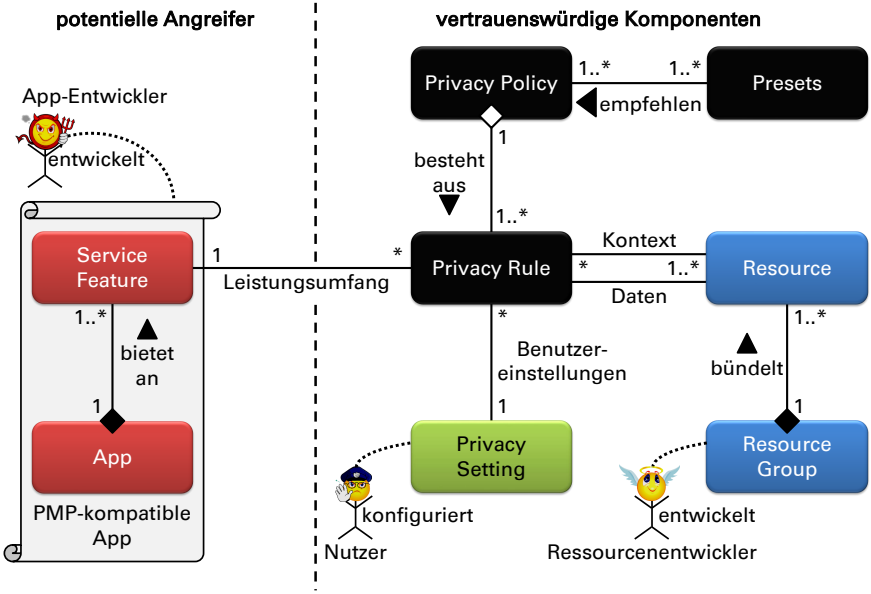


Abbildung 4.1: Schema des *PPMs* (in Anlehnung an [SM14])

auch die *PMP* davon aus, dass Bedrohungen stets von außen, d. h. durch Apps von Drittanbietern, ausgehen und nicht durch kompromittierte Komponenten des Systems selbst. Eine Erweiterung der *PMP*, die Schutzmaßnahmen gegen Angriffsvektoren von innen und von außen mit sich bringt, wird in Kapitel 5 vorgestellt.

4.1.1 Resources und Resource Groups

Das *PPM* führt sogenannte *Resources* ein, die dem Nutzer eine Schnittstelle zu geschützten Systemfunktionen und Daten liefern. Eine *Resource* kann dabei entweder einer konkreten Hardwarekomponenten (z. B. eine *Resource* zur Erfassung der Global Positioning System (GPS)-Daten) oder einer bestimmten Art von Daten (z. B. eine *Resource* für den Zugriff auf die Kontaktdaten) zugeordnet sein. Auch kann eine *Resource* Daten aus mehreren Quellen kombinieren und somit Apps höherwertige Dienste zur Verfügung stellen.

Je nach Berechtigung geben *Resources* ihre Daten vollständig, gefiltert, in aggregierter Form oder völlig randomisiert weiter. Je nach Datentyp stehen dabei unterschiedliche Filter, Aggregationsarten und Randomisierungsverfahren zur Verfügung. Beispielsweise kann eine *Resource* für Kontakte die Daten, die einer bestimmten Kategorie zugeordnet werden (z. B. „privat“) aus einer Anfrage herausfiltern, während eine *Resource* für Standortdaten die Genauigkeit der Positionsangabe auf x Meter herabsetzen kann. Die *Resource* selbst besitzt keine Zugriffsrechte auf die Daten, sondern bekommt diese nur kurzzeitig bei Bedarf von der Plattform, die das *PPM* implementiert.

Um in der Lage zu sein auch Hardwarekomponenten oder Datenquellen von zukünftigen Smart-Device-Generationen zu unterstützen, sind diese *Resources* selbst kein fester Bestandteil der Plattform, sondern können, ähnlich wie Treiber, der Plattform nachträglich hinzugefügt werden. Die *Resources* stammen von vertrauenswürdigen Drittanbietern, den *Resource*-Entwicklern.

Mehrere inhaltlich zusammengehörige *Resources* können in einer *Resource Group* zusammengefasst werden. Eine *Resource Group* beinhaltet die zur Installation nötigen Komponenten und stellt damit im Gegensatz zu den *Resources* eine installierbare Einheit dar. D. h. jede *Resource* muss mindestens einer *Resource Group* zugeordnet sein.

Informationen zu den *Resources* und zu den unterstützten Verfremdungstechniken werden zentral in der *Resource Group* hinterlegt. Im Fall der *PMP* werden diese Metadaten in der `rgis.xml`-Datei, dem sogenannten *Resource Group Information Set (RGIS)*, angegeben. Der Aufbau dieser Datei ist in Quelltext 4.1 ausschnitthaft abgebildet. Das vollständige Schema nach dem das *RGIS* aufgebaut ist, ist im Anhang in Quelltext A.1 hinterlegt.

Das `resourceGroupInformation`-Element beinhaltet Informationen, die für die Identifikation und Beschreibung der zugehörigen *Resource Group* nötig sind. Das Attribut `identifier` gibt einen eindeutigen Schlüssel an, über den die *Resource Group* eindeutig identifiziert und angesprochen werden kann. Daher wird hierfür der vollständige Pfad des Pakets, in dem sich die *Resource Group* befindet, verwendet. Jeder *Resource Group* ist ein Icon zugeordnet, das dem Nutzer angezeigt werden kann, damit er eine schnelle Übersicht über alle verfügbaren *Resource Groups* bekommt. Das `class`-Attribut gibt die

```
<?xml version="1.0" encoding="utf-8"?>
<resourceGroupInformationSet>
  <resourceGroupInformation
    identifier="Java-Package"
    icon="Pfad zum Icon"
    className="Java-Klassenname">
    <name lang="en">Resource-Name</name>
    ...
    <description
     ↪  lang="en">Resource-Beschreibung</description>
    ...
  </resourceGroupInformation>
  <privacySettings>
    <!- Liste der angebotenen Privacy Settings ->
  </privacySettings>
</resourceGroupInformationSet>
```

Quelltext 4.1: Exemplarischer Aufbau eines *RGISs*

Einstiegsklasse an, mit der die in der Gruppe enthaltenen *Resources* initialisiert werden können.

Während diese Informationen für das Schutzsystem gedacht sind, damit dieses die *Resource Groups* einbinden und die *Resources* ansprechen kann, beinhaltet das *RGIS* auch Metadaten, die dem Nutzer Informationen über die *Resource Group* geben. Das eingebettete `name`-Element legt den für den Nutzer sichtbaren Namen der *Resource Group* fest. Damit der Nutzer mehr Informationen über die *Resource Group* erhält und nicht wie bei den *Permissions* in Android mit einem oft kryptischen Bezeichner alleine gelassen wird, besitzt jede *Resource Group* auch ein `description`-Element, das eine beliebig lange natürlichsprachliche Beschreibung der Funktionen der jeweiligen *Resource Group* enthält.

Um die Internationalisierung der *Resource Groups* zu vereinfachen, kann sowohl der Name als auch die Beschreibung der *Resource Group* in beliebig vielen Sprachen angegeben werden. Hierfür muss lediglich das `lang`-Attribut gemäß ISO 3166[2] gesetzt werden. Mindestens müssen die Angaben jedoch auf

[2] siehe ISO [ISO13a]

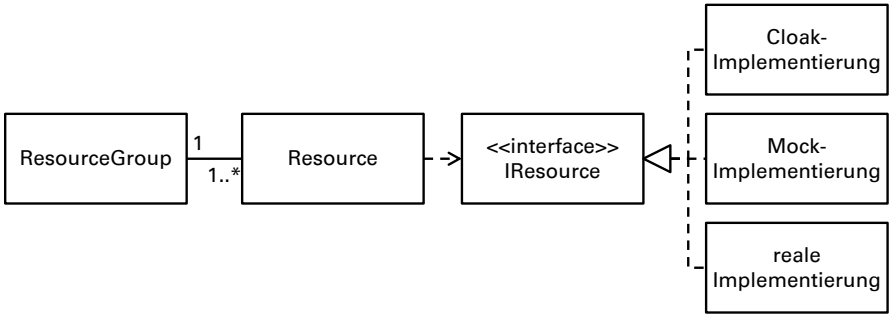


Abbildung 4.2: Modell der *Resource Groups*

Englisch vorhanden sein. Auf den Aufbau und Inhalt des `privacySettings`-Elements wird in Abschnitt 4.1.2 näher eingegangen.

Der Aufbau einer *Resource Group* ist in Abbildung 4.2 vereinfacht dargestellt. Aus Implementierungssicht handelt es sich bei einer *Resource Group* um einen Android-Service, d. h. eine installierbare und ausführbare Android-Komponente ohne eigenes Graphical User Interface (GUI). Jede *Resource Group* besitzt eine Hauptklasse (abgeleitet von einer generischen Vaterklasse `ResourceGroup`), die die *Resources* gemäß den Angaben aus dem *RGIS* bei der zugrundeliegenden Plattform registriert. Über diese Hauptklasse lassen sich alle enthaltenen *Resources* identifizieren und ansprechen. Eine *Resource Group* kann beliebig viele *Resources* beinhalten – man denke beispielsweise an eine *Resource Group* zur Ermittlung des aktuellen Standorts, die diesen u. a. über GPS, das Funknetz oder das Wireless Local Area Network (WLAN) beziehen kann – muss allerdings mindestens eine *Resource* enthalten.

Für jede *Resource*, die sich von einer generischen Vaterklasse `Resource` ableitet, muss eine öffentliche Schnittstelle angegeben werden, welche über die Dienste, die von der jeweiligen *Resource* angeboten werden, informiert. Im Fall von Android bietet sich hierfür Android Interface Definition Language (AIDL) an, eine stark an Java angelehnte Sprache zur Definition von Schnittstellen[3]. Nur über die in dieser Schnittstelle definierten Methoden kann ein Datenaustausch stattfinden. In Quelltext 4.2 ist exemplarisch ein Auszug aus einer Schnittstellendefinition für eine GPS-*Resource* angegeben.

[3] siehe https://developer.android.com/guide/components/aidl.html

```
1 package
  ↪ de.unistuttgart.ipvs.pmp.resourcegroups.location.aidl;
2
3 interface IGPSResource {
4    boolean isGpsEnabled();
5    void startLocationLookup(in long minTime, in float
      ↪ minDistance);
6    void endLocationLookup();
7    boolean isLocationUpdateAvailable();
8    double getLongitude();
9    double getLatitude();
10   float getAccuracy();
11   String getAddress();
12 }
```

Quelltext 4.2: Auszug aus einer Schnittstellendefinition für eine GPS-*Resource* in AIDL

Dabei unterstützt die *Resource* genau wie der GPS-*LocationProvider* von Android die dauerhafte Überprüfung des Standorts sowie die Abfrage des Längen- und Breitengrads. Zusätzlich ist es aber auch möglich höherwertige Daten abzufragen. Hierzu zählt beispielsweise die Abbildung der Koordinaten auf eine Adresse. Außerdem können die Daten stark aggregiert (z. B. auf Landesebene) angefragt werden, um die privaten Daten, d. h. den exakten Standort des Nutzers, nicht an eine App weiterzugeben. Wie der Nutzer festlegen kann, welche dieser Funktionen von einer App genutzt werden dürfen und wie er die Datenqualität weiter reduzieren kann, um noch weniger private Daten preiszugeben, ist in Abschnitt 4.1.2 beschrieben.

Wie in Abbildung 4.2 zu sehen ist, existiert je *Resource* allerdings nicht nur eine Implementierung dieser Schnittstelle, sondern jeweils drei. Neben der realen Implementierung, die die Funktionen vollständig implementiert und die korrekten Daten zurückliefert[4], gib es zwei weitere Implementierungen, die für jede Funktion Zufallswerte zurückliefern. Der Nutzer kann für jede *Resource* und *App* selbst entscheiden, welche der drei Implementierungen angesprochen

[4] Evtl. bedürfen die definierten *Privacy Settings* (siehe Abschnitt 4.1.2) einer nachträglichen Anpassung der Daten.

werden soll. Der Unterschied zwischen der *Mock*-Implementierung und der *Cloak*-Implementierung ist, dass die *Mock*-Implementierung einer App mitteilt, dass die übermittelten Werte randomisiert sind und es der App überlässt, wie sie demzufolge mit den Daten umgeht. Bei der *Cloak*-Implementierung ist dies der App nicht bekannt. Die Randomisierungsalgorithmen, die in der *Cloak*-Implementierung zum Einsatz kommen, sind häufig wesentlich komplexer und damit rechenintensiver sind als die der *Mock*-Implementierung. Betrachtet man beispielsweise die GPS-*Resource*, so reicht es für die *Mock*-Implementierung vollkommen aus, wenn man dafür sorgt, dass die Werte für den Längen- und Breitengrad innerhalb des jeweils gültigen Wertebereichs liegen. Soll einer App aber verschleiert werden, dass die übermittelten Daten verfälscht wurden, muss man zusätzlich dafür sorgen, dass die Werte stets realistisch sind. Es muss beispielsweise verhindert werden, dass sich der Standort des Nutzers innerhalb von Sekunden über eine große Distanz verändert. Eine App würde ansonsten ebenfalls bemerken, dass die übermittelten Daten verfälscht wurden und ggf. den von ihr angebotenen Service einschränken. Die Randomisierungstechniken unterscheiden sich dabei von *Resource* zu *Resource* sehr stark und müssen auf den jeweiligen Kontext angepasst sein.

4.1.2 Privacy Settings

Der Nutzer nimmt allerdings nicht nur mit der Auswahl der für seine Wünsche passenden Implementierung Einfluss auf die Daten (und Datenqualität), die eine App von einer *Resource* erhält. Jede *Resource Group* definiert darüber hinaus im *RGIS* die sogenannten *Privacy Settings*. Die *Privacy Settings* beschreiben, welche Berechtigungen im Bezug auf die jeweilige *Resource Group* vom Nutzer vergeben respektive von den Apps angefordert werden können und welche Einschränkungen bezüglich der Datenqualität vorgenommen werden können. Quelltext 4.3 zeigt einen Ausschnitt der Definition der *Privacy Settings* für eine GPS-*Resource*. Dieser Auszug müsste an der Stelle des *RGISs* eingefügt werden, die in Quelltext 4.1 ausgespart wurde. Alle Informationen können zwecks Internationalisierung in mehreren Sprachen (mindestens jedoch in Englisch) angegeben werden.

Jedes *Privacy Setting* wird über das `identifier`-Attribut adressiert, weshalb dieser Bezeichner innerhalb einer *Resource Group* eindeutig sein muss. Das

```
...
<privacySetting identifier="useLocation"
  ↪ validValueDescription="'true', 'false'">
  <name lang="en">GPS Location</name>
  <description lang="en">Obtains access to the exact current
    ↪ location</description>
  <changeDescription lang="en">Assigned apps are allowed to
    ↪ access GPS data.</changeDescription>
</privacySetting>
<privacySetting identifier="useLocationDescription"
  ↪ validValueDescription="'NONE', 'COUNTRY', 'CITY',
  ↪ 'STREET'">
  <name lang="en">Description Level</name>
  <description lang="en">Location description is more or less
    ↪ aggregated</description>
  <changeDescription lang="en">Defines a level of detail for
    ↪ the location description.</changeDescription>
</privacySetting>
<privacySetting identifier="locationAccuracy"
  ↪ validValueDescription="positive integer">
  <name lang="en">Accuracy</name>
  <description lang="en">The maximal accuracy of the location
    ↪ data</description>
  <changeDescription lang="en">A value to define the maximum
    ↪ accuracy in meters.</changeDescription>
</privacySetting>
...
```

Quelltext 4.3: Drei exemplarische *Privacy Settings* für eine GPS-*Resource*

zweite Attribut (`validValueDescription`) des *Privacy Settings* definiert den Wertebereich, der für diese *Privacy Setting* zulässig ist. Dabei handelt es sich um eine für Menschen lesbare Beschreibung, die nicht automatisch ausgewertet, sondern lediglich dem Nutzer bei der Konfiguration als Hilfestellung angezeigt wird. Die eigentliche Festlegung auf einen konkreten Datentyp erfolgt innerhalb der `ResourceGroup`-Klasse, wenn die Einträge der *RGIS* ausgewertet und die Komponenten registriert werden.

Die drei im `privacySetting`-Element eingebetteten Elemente (`name`, `description` und `changeDescription`) dienen alle ebenfalls dem Nutzer nur als Beschreibung des *Privacy Settings*. So kann damit ein verständlicher Name vergeben werden sowie beschrieben werden, welche Berechtigungen eine *Resource* durch das jeweilige *Privacy Setting* erhält sowie welche Auswirkungen Änderungen an dem *Privacy Setting* mit sich bringen.

Damit die *Privacy Settings* genutzt werden können, muss der *Resource*-Entwickler diese zunächst registrieren und einem Datentyp zuordnen. In Quelltext 4.3 sind drei unterschiedliche Typen von *Privacy Settings* gezeigt, die unmittelbar von der *PMP* unterstützt werden. Der einfachste Typ, das `BooleanPrivacySetting`, verwendet ausschließlich Wahrheitswerte, womit der Nutzer einzelne Funktionen ein- bzw. ausschalten kann. Dies entspricht noch am ehesten den *Permissions* in Android, wobei die Berechtigungsvergabe im *PPM* wesentlich feingranularer erfolgen kann. Das lässt sich am Besten am Beispiel des *Privacy Settings* `useLocation` demonstrieren. Unter Android benötigt eine App die `ACCESS_FINE_LOCATION`-*Permission*, um auf den Standort eines Nutzers zuzugreifen. Erhält sie jedoch diese *Permission*, so kann sie nicht nur auf den Standort des Nutzers zugreifen, sondern kann auch die aktuelle Bewegungsgeschwindigkeit, die Ausrichtung des Smart Devices und selbst die Laufzeit des Geräts seit dem Systemstart erfragen, da all diese Aktivitäten unter der gleichen *Permission* subsumiert werden, obwohl sie wenig bis nichts mit dem Standort zu tun haben. Im *PPM* kann der *Resource*-Entwickler hingegen für jede einzelne Funktion (und damit auch für die entsprechenden Daten), die in der *Resource*-Schnittstelle definiert wird, separat festlegen, welche *Privacy Settings* dafür nötig sind.

Während diese Eigenschaft des *PPMs* bereits einen Vorteil gegenüber den Android *Permissions* darstellt, gehen die beiden anderen *Privacy-Settings*-Typen noch wesentlich weiter. Das `EnumPrivacySetting` gibt dem Nutzer eine Auswahl diskreter Wert vor. Auf diesen Werten existiert eine Ordnung[5]. Auf diese Weise kann ein Nutzer einer App stufenweise Zugang zu seinen Daten gewähren, bzw. die Genauigkeit oder die Qualität der Daten stufenweise reduzieren respektive erhöhen. Am Beispiel des `useLocationDescription`-*Privacy-*

[5] Technisch wird diese Ordnung dadurch ermöglicht, dass der Wertebereich eines Aufzählungstyps unter Java auf einen ganzzahligen Identifikator abgebildet wird.

Settings lässt sich dies verdeutlichen. Die zulässigen Werte, zwischen denen ein Nutzer wählen kann erlauben es einer App entweder keine Informationen über seinen Standort zu geben (`NONE`) oder das Land (`COUNTRY`), die Stadt (`CITY`) oder sogar die vollständige Adresse (`STREET`) mit der App zu teilen. In diesem Beispiel schließen größere Werte (größer ist hierbei auf die Ordnungsrelation bezogen) die darunterliegenden Werte stets mit ein. Dies ist allerdings optional und dem *Resource*-Entwickler überlassen, ob er die vorhandene Ordnung nutzt oder die Einstellungsmöglichkeiten vollständig disjunkt zueinander sind. Dieses *Privacy Settings* ist unabhängig von den anderen *Privacy Settings*, d. h. einer App können aggregierte Daten zu dem Standort gegeben werden, ohne dass diese App Zugriff auf die konkreten GPS-Daten erhält. Eine *Resource* kann die *Privacy Settings* stets auswerten und einer App entsprechend der Einstellungen, die für diese App gelten, die angefragten Daten aufbereiten. Die `getAddress`-Methode gibt beispielsweise immer einen String zurück, der die Adresse auf dem vom Nutzer freigegebenen Level enthält (ggf. kann dieser String leer sein, falls der Nutzer die `NONE`-Option gewählt hat). D. h. Apps können diese Methode immer verwenden, ohne sich um die Nutzereinstellungen kümmern zu müssen. Wie Apps im *PPM* einen Mindestlevel bezüglich der von ihnen benötigten Datenqualität definieren und wie sie mit einer reduzierten Datenqualität umgehen können, ist in Abschnitt 4.1.3 beschrieben.

Der letzte *Privacy-Settings*-Typ ist das `IntegerPrivacySetting`. Hier kann der Nutzer jeden beliebigen ganzzahligen Wert[6] angeben. Über das `locationAccuracy`-*Privacy-Settings* kann der Nutzer beispielsweise die Genauigkeit, mit dem sein Standort einer App zur Verfügung gestellt wird, festlegen. Diese Einstellung hat damit unmittelbaren Einfluss auf sämtliche Standortanfragen, die von der GPS- *Resource* angeboten werden. Während die *Resource* auch weiterhin alle Daten in der bestmöglichen Qualität erhält, wird sie bei jeder Anfrage einer App darüber informiert, welche Einstellungen der Nutzer für diese App festgelegt hat, und muss die Daten vor der Weitergabe entsprechend verfälschen. Im Fall des Standorts bedeutet dies, dass x Meter auf die tatsächliche Position addiert bzw. subtrahiert werden. Auch hierfür kann eine App einen Mindestlevel definieren, der für die Ausführung der App erforderlich ist.

[6] Implementierungsbedingt sind wegen der von Java vorgegebenen Restriktionen nur Werte im Intervall $[-2^{31},\ 2^{31}-1]$ zulässig.

So kann beispielsweise eine Navigations-App nur dann korrekt funktionieren, wenn die angegebenen Standortdaten nicht zu sehr von dem tatsächlichen Standort des Nutzers abweichen. Wie mit dieser Problematik im Fall des *PPMs* umgegangen wird, ist in Abschnitt 4.1.3 beschrieben.

Weitere *Privacy-Settings*-Typen können für andere *Resources* sinnvoll sein. So könnte man sich beispielsweise einen String-basierten Typ für eine Internet-*Resource* vorstellen, mit dem der Nutzer einzelne Internetadressen für eine App freigibt respektive bestimmte Adressen für eine App sperrt. Auch können andere *Resources* die Angabe von reellen Zahlen erforderlich machen, um dem Nutzer feingranularere Einstellungen zu ermöglichen. Dies sind nur zwei von vielen möglichen *Privacy-Settings*-Typen. Die modulare Struktur, mit dem das *PPM* und die *PMP* aufgebaut sind, erlaubt es *Resource*-Entwicklern beliebige eigene Typen zu definieren und in ihren *Resources* bzw. *Privacy Settings* zu verwenden. Die Auswahl, welche *Privacy Settings* sinnvoll sind und wie sich die Einstellungen des Nutzers auf die Datenqualität auswirken, ist stark vom Kontext der jeweiligen *Resource* abhängig und muss daher vom *Resource*-Entwickler von Fall zu Fall entschieden werden.

4.1.3 Apps und Service Features

Durch das Konzept der *Resources* und der *Privacy Settings* wird der Nutzer in die Lage versetzt, die Zugriffsrechte von Apps auf Systemfunktionen und Daten feingranular zu reglementieren. Dies bringt allerdings zwei Nachteile mit sich. Einerseits sind sich viele Nutzer gar nicht darüber bewusst, welche Einstellungen für ihre Bedürfnisse geeignet sind. Andererseits benötigen Apps bestimmte Funktionen oder Daten zur Realisierung ihrer angedachten Aufgaben. Daher benötigt der Nutzer mehr Informationen darüber, welche Berechtigungen er einer App geben sollte, um den von ihm gewünschten Funktionsumfang zu erhalten. Darüber hinaus müssen auch die Apps wesentlich flexibler werden, damit sie auch mit reduzierten Berechtigungen umgehen können.

Daher führt das *PPM* zusätzlich zu Apps sogenannte *Service Features* ein. Jede App, die kompatibel zum *PPM* ist, muss definieren, welche Daten und Funktionen, die von *Resources* angeboten werden, sie benötigt und welchen Funktionsumfang sie damit dem Nutzer anbieten kann. Eine solche Definition wird als *Service Feature* bezeichnet. Jede App kann ein oder mehrere *Service*

```
<?xml version="1.0" encoding="UTF-8"?>
<appInformationSet>
  <appInformation>
    <name lang="en">App-Name</name>
    ...
    <description lang="en">App-Beschreibung</description>
    ...
  </appInformation>
  <serviceFeatures>
    <!- Liste der angebotenen Service Features ->
  </serviceFeatures>
</appInformationSet>
```

Quelltext 4.4: Exemplarischer Aufbau eines *AISs*

Features anbieten. Berechtigungen werden demnach nicht für Apps vergeben, sondern für deren *Service Features*. Diese *Service Features* können beliebig ein- bzw. ausgeschaltet werden, wodurch der Funktionsumfang der App an die Bedürfnisse des Nutzers angepasst werden kann. Je weniger *Service Features* vom Nutzer benötigt werden, desto weniger Daten gibt er preis.

Analog zu den *Resource Groups* müssen auch Apps Metadaten angeben, die die App sowie deren *Service Features* beschreiben. Bei der *PMP* erfolgt dies ebenfalls in einer Extensible Markup Language (XML)-Datei, der `ais.xml`. Das Schema für dieses sogenannte *Application Information Set (AIS)* ist im Anhang in Quelltext A.2 hinterlegt. Wie in Quelltext 4.4 zu sehen ist, ist der Grundaufbau dieser Datei sehr ähnlich zum Aufbau der RGIS. Auch beim AIS können zwecks Internationalisierung alle Informationen in mehreren Sprachen (mindestens jedoch in Englisch) angegeben werden.

Zunächst werden im ersten Element des *AISs* (`appInformation`) Angaben zur übergeordneten Komponente, in diesem Fall der App, getätigt. Hierzu gehört neben einem Bezeichner für die App (`name`) eine natürlichsprachliche Beschreibung (`description`), welchen Zweck die App erfüllt. Beide Angaben sind ausschließlich für den Nutzer gedacht. Identifikatoren, wie sie im *RGIS* erforderlich sind, werden im *AIS* nicht benötigt. Da es sich aus Implementierungssicht um eine normale Android-App handelt, die lediglich um

```
...
<serviceFeature identifier="showLocation">
  <name lang="en">GPS-based Location</name>
  <description lang="en">Show current location.</description>
  <changeDescription lang="en">This Service Feature uses the
   ↪ GPS to show your current location on a
   ↪ map.</changeDescription>
  <requiredResourceGroup identifier="location"
   ↪ minRevision="1">
    <requiredPrivacySetting identifier="useLocation">
      true
    </requiredPrivacySetting>
    <requiredPrivacySetting identifier="locationAccuracy">
      50
    </requiredPrivacySetting>
  </requiredResourceGroup>
</serviceFeature>
...
```

Quelltext 4.5: Exemplarisches *Service Feature* für eine Navigations-App

zusätzliche Metadaten angereichert wurde, bringt diese bereits alle benötigten Identifikatoren mit.

Im zweiten Element des *AISs* dem `serviceFeatures`-Element, werden anschließend die *Service Features* definiert. Ein exemplarisches *Service Feature* wird in Quelltext 4.5 angegeben. Es beschreibt ein mögliches *Service Feature* einer Navigations-App, das dafür sorgt, dass der aktuelle Standort des Nutzers auf einer Karte angezeigt wird.

Damit die *Service Features* eindeutig identifiziert und angesprochen werden können, muss jedem `serviceFeature`-Element im Attribut `identifier` ein Bezeichner zugewiesen werden. Dieser Bezeichner muss innerhalb einer App eindeutig sein. Der eigentlich verwendete Identifikator setzt sich allerdings aus dem Paketnamen der App und diesem Bezeichner zusammen, d. h. unterschiedliche Apps können die gleichen Bezeichner für ihre *Service Features* verwenden. Anschließend folgen analog zur Definition der *Privacy Settings* drei Elemente (`name`, `description` und `changeDescription`), die der Beschreibung des *Service Features* für den Nutzer dienen. Mithilfe dieser Informationen ist der

Nutzer in der Lage zu entscheiden, welche *Service Features* er nutzen möchte, bzw. welche Einschränkungen er in Kauf nehmen muss, wenn er der App die für das *Service Feature* benötigten Zugriffsrechte entzieht.

Das `requiredResourceGroup`-Element stellt den wesentlichen Unterschied zur Definition der *Privacy Settings* dar. In diesem Element wird beschrieben, welche *Resource Groups* auf dem Gerät verfügbar sein müssen, damit das *Service Feature* überhaupt nutzbar ist. Da *Resource Groups* dynamisch vom Nutzer installiert bzw. deinstalliert werden können, ist es erforderlich sicherzustellen, dass alle für ein *Service Feature* benötigten *Resource Groups* vorhanden sind. Eine App kann auch ausgeführt werden, wenn nicht alle im *AIS* angegebenen *Resource Groups* installiert sind, jedoch können die dazugehörigen *Service Features* nicht genutzt werden. Im `identifier`-Attribut wird auf den Identifikator einer *Resource Group* verwiesen, wie er im *RGIS* angegeben ist. Zusätzlich kann eine Mindestversionsnummer der *Resource Group* explizit angegeben werden (`minRevision`). Da *Resource Groups* ständig weiterentwickelt werden und deren Funktionsumfang stetig ausgeweitet wird, kann somit sichergestellt werden, dass der Nutzer bei sich die richtige Version installiert hat.

Innerhalb jedes `requiredResourceGroup`-Elements werden die zur Ausführung des *Service Features* benötigten *Privacy Settings* beschrieben. Jedes `requiredPrivacySetting`-Element gibt zunächst den Identifikator des angeforderten *Privacy Settings* an (im Attribut `identifier`) und definiert darin welchen Wert dieses *Privacy Setting* mindestens haben muss. Bei den *PrivacySettings*-Typen `EnumPrivacySetting` und `IntegerPrivacySetting` können die *Privacy Settings* daher auch weniger restriktiv sein als der im *Service Feature* angegebene Wert. Ggf. kann dadurch die Servicequalität der App gesteigert werden.

Dies lässt sich an dem in Quelltext 4.5 gegebenen Beispiel leicht verdeutlichen. Die Navigations-App ist u. a. in der Lage den Standort des Nutzer auf einer Karte anzuzeigen. Hierfür benötigt das entsprechende *Service Feature* der App Zugriff auf die GPS-Daten. Dieser Zugriff erfolgt über die `location`-*ResourceGroup*. Der Nutzer muss zustimmen, dass dieses *Service Feature* das boolesche *Privacy Setting* `useLocation` aktiviert und die Genauigkeit der Standortdaten um maximal 50 Meter von der entsprechenden *Resource* verfälscht werden (`locationAccuracy`). Je weniger der Nutzer die Standortdaten verfälscht, des-

to exakter kann seine Position angezeigt werden. Ist der Nutzer nicht damit einverstanden, dass die Navigations-App diese Daten erhält, so kann er dieses *Service Feature* deaktivieren, die restlichen Funktionen der App aber dennoch nutzen. Z. B. könnte die App auch ohne diese Daten eine statische Karte mit lediglich den Zielkoordinaten anzeigen.

Besitzt eine App mehrere *Service Features*, so können zwei (oder mehr) dieser *Service Features* SF_1 und SF_2 identische *Privacy Settings* anfordern. Der Nutzer kann dennoch SF_1 und SF_2 unabhängig voneinander aktivieren respektive deaktivieren. Es findet innerhalb einer App keine Weitergabe der Daten zwischen *Service Features* statt. D. h. der Nutzer kann beispielsweise SF_1 Zugriff auf den Standort mit einer Genauigkeit von 10 Meter gewähren, während SF_2 diese Daten nur auf 100 Meter genau erhält.

Die Logik der *Service Features* wird im App-Code realisiert. Bevor der Code eines *Service Features* ausgeführt werden kann, muss vom Entwickler geprüft werden, ob das *Service Feature* vom Nutzer aktiviert wurde[7]. Hierfür stellt die Plattform, die das *PPM* implementiert (z. B. die *PMP*), entsprechende Funktionen zur Verfügung, die alle benötigten Überprüfungen durchführen und ggf. dem Nutzer Informations- und Konfigurationsdialoge anzeigen. Damit eine App diese Funktionen nutzen kann, muss sie sich zunächst bei der Plattform registrieren. In diesem Prozess werden die Informationen aus dem *AIS* eingelesen und verarbeitet. Alle für die Registrierung benötigten Informationen können direkt aus der App ausgelesen werden (im Fall von Android aus dem *App Manifest*).

4.1.4 Privacy Rule und Privacy Policy

Während die obigen Bestandteile des *PPMs* von physischen Softwarekomponenten realisiert werden, handelt es sich bei einer *Privacy Rule* und damit auch der kompletten *Privacy Policy* um virtuelle Komponenten. Eine *Privacy Rule* stellt einen Knotenpunkt dar, der die *Service Features* mit vom Nutzer definierten *Privacy Settings* und Daten aus den *Resources* miteinander verknüpft.

In einer *Privacy Rule* wird beschrieben, welche Daten und Funktionen eine App zur Erbringung einer Funktionalität nutzen möchte (definiert im *Service*

[7] Seit Android 6.0 ist eine solche Überprüfung beim Zugriff auf bestimmte Daten erforderlich, jedoch ohne das Potenzial des *PPMs* zu bieten (siehe Abschnitt 2.3).

Feature) und welche Berechtigungen der Nutzer ihr dafür gewährt (festgelegt im *Privacy Setting*). Dies muss nicht zwangsläufig deckungsgleich sein. So kann der Nutzer einer App mehr Berechtigungen einräumen, als diese anfordert, um eine bessere Servicequalität zu erhalten (siehe das Beispiel der Navigations-App in Abschnitt 4.1.3). Außerdem kann der Nutzer auch weniger Berechtigungen gewähren (mit der Konsequenz, dass das *Service Feature* deaktiviert wird) oder eine *Mock*-Implementierung bzw. *Cloak*-Implementierung anstelle der realen Implementierung der *Resource* verwenden, um eine App ausschließlich mit randomisierten Daten zu versorgen.

Darüber hinaus kann der Nutzer in einer *Privacy Rule* auch Daten aus den *Resources* via Kontextannotationen miteinbeziehen, die die Gültigkeit der Regeln einschränken können, während eine *Privacy Rule* ohne Annotation dauerhaft gültig ist. Auf diese Weise kann der Nutzer *Privacy Rules* angeben, die die Berechtigungen der Apps respektive deren *Service Features* beispielsweise während der Arbeitszeit stark einschränken und in seiner Freizeit ausweiten. Hierzu muss er lediglich die Situation „Arbeit“ und „Freizeit“ auf Basis von *Resource*-Werten beschreiben; der Wechsel zwischen „Arbeit“ und „Freizeit“ erfolgt daraufhin automatisch und das System erkennt, welche *Privacy Rule* angewandt werden muss. Wie diese Konfiguration durchgeführt wird, ist in Abschnitt 4.2.6 näher beschrieben.

Die *Privacy Policy* speichert die Gesamtheit aller *Privacy Rules* eines Nutzers. Das hierfür verwendete relationale Datenbankschema ist in Abbildung 4.3 dargestellt. Die *Service Features* und *Privacy Settings* werden darin durch ihren Namen sowie den Paketnamen der App bzw. der *Resource Group*, in der sie definiert wurden, identifiziert[8]. Obwohl diese Kombination bereits ein Schlüsselkandidat ist, wird als Primärschlüssel jeweils ein ganzzahliger Identifikator (`_id`) eingeführt, um leichter auf die Einträge verweisen zu können.

In der Tabelle `SF_Required_PS` ist hinterlegt, welche Funktionen und Daten eine App nutzen will, während die Tabelle `GrantedValues` beschreibt, welche Zugriffe der Nutzer erlaubt hat. Die Auswertung, welche *Service Features* unter den gewährten Berechtigungen nutzbar sind, erfolgt dynamisch zur Laufzeit. Dies ist einerseits erforderlich, damit die Berechtigungen zur Laufzeit geändert

[8] Der Paketname wird benötigt, da die Namen der *Service Features* und *Privacy Settings* nur innerhalb einer App bzw. *Resource Group* eindeutig ist.

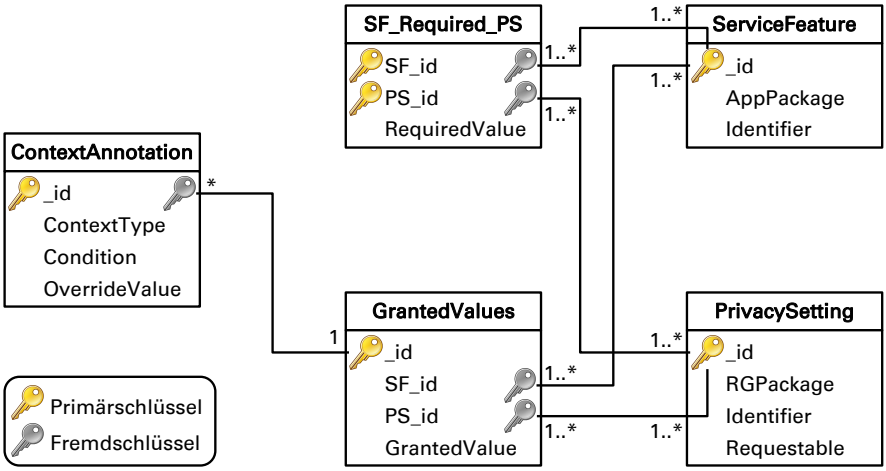


Abbildung 4.3: Datenbankschema der *Privacy Policy*

werden können[9] (siehe Abschnitt 4.2.2), und andererseits, um den Kontext des Nutzers bei der Auswertung berücksichtigen zu können (siehe Abschnitt 4.2.6). Diese Kontextannotationen werden in der Tabelle `ContextAnnotation` hinterlegt. Der Kontext wird über seinen Typ (`ContextType`) beschrieben. Der Typ kann beispielsweise der Standort oder die Uhrzeit respektive das Datum sein. Grundsätzlich kann jede Art von Daten verwendet werden, auf die über *Resources* zugegriffen werden kann. Im Eintrag `Condition` wird beschrieben, wann, d. h. bei welchem Datenwert, der beschriebene Kontext vorliegt. Falls der Kontext eintritt, so gibt `OverrideValue` an, welchen Wert in diesem Fall das zugehörige *Privacy Setting* annehmen soll. Die *Privacy Policy* sowie jede beliebige Teilmenge der *Privacy Rules* kann als *Preset* ex- bzw. importiert werden, um diese zwischen unterschiedlichen Geräten auszutauschen. Mehr Details zu den *Presets* sind in Abschnitt 4.2.7 gegeben.

[9] Ein *Service Feature* kann eine Vielzahl an benötigten *Privacy Settings* definieren und erst die Kombination dieser Werte entscheidet darüber, ob das *Service Feature* aktivierbar ist. Die Auswertung muss daher nach der Änderung jedes *Privacy Settings* erneut erfolgen.

4.2 Eigenschaften der PMP

Die *PMP* stellt eine Implementierung des *PPMs* dar. Da es beim *PPM* im wesentlichen um die feingranulare und flexible Vergabe von Zugriffsberechtigungen geht, ist die *PMP* primär ein Berechtigungssystem und sorgt damit für die Einhaltung der Datenschutzrichtlinien, gemäß der Definition aus Abschnitt 1.2.2. Allerdings erfüllen die speziellen Eigenschaften der *PMP* bereits einige der Schutzziele der Datensicherheit (siehe Abschnitt 1.3). Eine genaue Evaluation, inwiefern die aufgestellten Schutzziele erfüllt werden, ist in Abschnitt 7.1 gegeben. An dieser Stelle soll lediglich die *PMP* vorgestellt werden, so dass sie hinsichtlich ihrer Charakteristika mit anderen verwandten Arbeiten verglichen werden kann (siehe Abschnitt 4.5).

Der Arbeitsablauf der *PMP* ist vereinfacht in Abbildung 4.4 dargestellt. Wenn ein Nutzer eine neue App installiert (N), so wird die Installation zunächst gemäß den Vorgaben des verwendeten Betriebssystems durchgeführt (1). War die Installation erfolgreich, registriert sich die App automatisch bei der *PMP* und übermittelt ihr *AIS* (2). Die *PMP* liest daraus die *Service Features* sowie die dafür benötigten *Privacy Settings* aus und gibt diese Information an den Nutzer zurück (F). Der Anwender erstellt eine Konfiguration der *Privacy Settings*, indem er für ihn relevante *Service Features* aktiviert. Dadurch entsteht ein initialer Satz *Privacy Rules*, die in der *Privacy Policy* abgelegt werden (3). Sind für die Ausführung der ausgewählten *Service Features* weitere Resource Groups nötig, so können diese von der *PMP* bedarfsgerecht nachinstalliert werden (4) (siehe Abschnitt 4.2.5).

Der Zugriff auf Daten der *Resources* erfolgt stets über die *PMP*. Hierfür greift eine App auf die öffentlichen Schnittstellen der *Resources* zu. Diese Anfrage geht zunächst an die *PMP* (5), die in der *Privacy Policy* nachschaut, ob es darin für diese App und diese Anfrage eine *Privacy Rule* gibt (6). Ist dies der Fall, wird abhängig von dieser *Privacy Rule* der Zugriff auf die Schnittstelle (und damit indirekt auch auf die System- und Nutzerdaten) erlaubt respektive verboten (7). Gibt es noch keine entsprechende *Privacy Rule*, so wird der Nutzer darüber informiert, dass ein *Service Feature* aufgrund von fehlenden Berechtigungen nicht ausgeführt werden kann. Der Nutzer hat daraufhin die Möglichkeit, die *Privacy Settings* entsprechend seiner Anforderungen zu

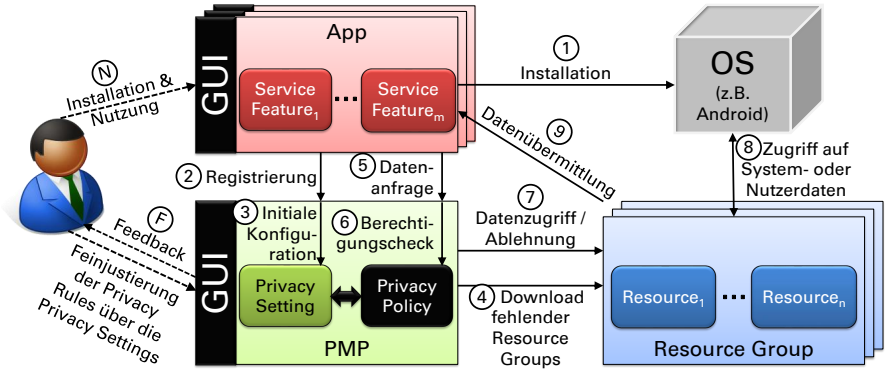
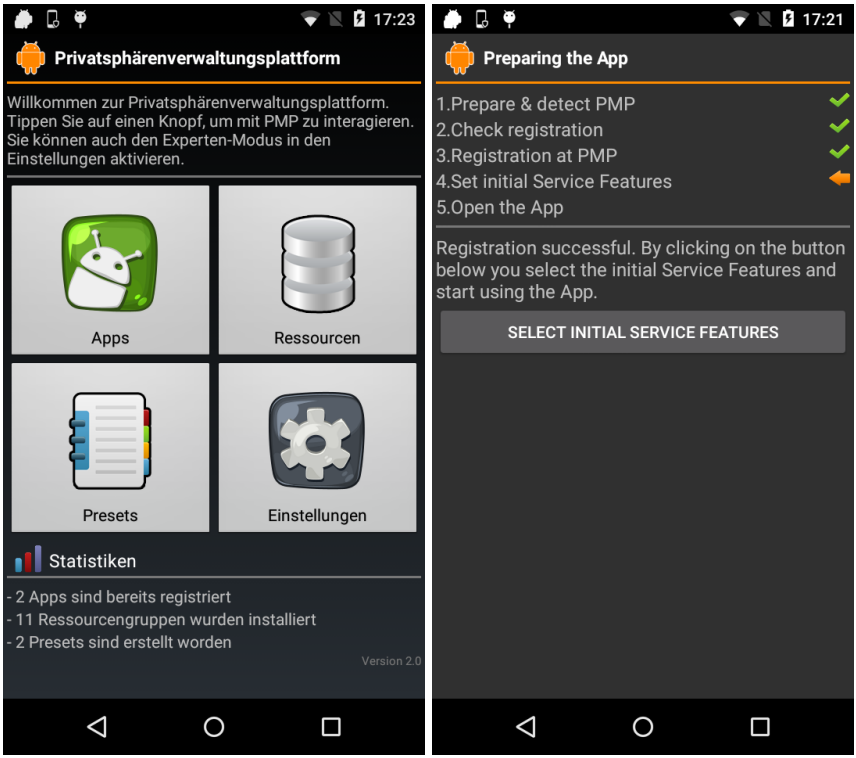


Abbildung 4.4: Schematischer Arbeitsablauf der *PMP* (in Anlehnung an [SM15])

konfigurieren (F). Wird der *Resource*-Zugriff gestattet, so sammelt die *Resource* die angefragten Daten von den jeweiligen Quellen ein (8), führt entsprechend der *Privacy Settings*, die nötigen Verfremdungen oder Aggregationsfunktionen durch und leitet die Daten abschließend an die anfragende App weiter (9).

Während ein Großteil der Programmlogik für den Nutzer verborgen bleibt und im Hintergrund dafür sorgt, dass die von ihm in der *Privacy Policy* konfigurierten *Privacy Rules* eingehalten werden (Details hierzu in Abschnitt 4.3 bzw. Abschnitt 4.4), bietet die *PMP* ihm auch eine GUI und mehrere Dialogfenster, mit denen er die *Privacy Rules* definieren kann. Abbildung 4.5a zeigt das Hauptmenü der *PMP*. Über dieses Hauptmenü hat der Nutzer direkten Zugriff auf alle Apps, die von der *PMP* verwaltet werden, und auf alle *Resource Groups*, die bislang der *PMP* hinzugefügt wurden. Darüber hinaus kann er seine *Presets* verwalten (siehe Abschnitt 4.2.7) sowie Einstellungen am System vornehmen. Da sich die *PMP* sowohl an erfahrene Nutzer als auch an Laien richtet, ist es u. a. möglich, zwischen einem einfachen Modus und einem Expertenmodus zu wechseln. Der einfach Modus reduziert die vom Nutzer zu fällenden Entscheidungen im Wesentlichen darauf, welche *Service Features* er aktivieren möchte, während der Expertenmodus ihm eine feingranulare Konfiguration der *Privacy Settings* gestattet. Details hierzu finden sich in Abschnitt 4.2.2. Außerdem können in den Einstellungen sehr viele Ereignisprotokolle erstellt werden. Die



(a) Hauptbildschirm der *PMP* (b) Installationsdialog der *PMP*

Abbildung 4.5: GUI der *PMP*

PMP verzeichnet jedwede Änderung am System bzw. der *Privacy Policy* und kann diese so bei Bedarf in eine für den Nutzer lesbare Form aufbereiten.

Wie in Abschnitt 4.1.3 beschrieben, muss sich eine App während der Installation bei der *PMP* registrieren und u. a. ihre *Service Features* offenlegen. Wenn dies geschieht, bekommt der Nutzer den *PMP*-Installationsdialog (Abbildung 4.5b) angezeigt. Der Dialog zeigt die für die Registrierung nötigen Schritte an und informiert über eventuelle Fehler, die in diesem Prozess auftreten. Zusätzlich kann der Nutzer über den *PMP*-Installationsdialog einen ersten Satz *Privacy Rules* für die installierte App erstellen. Per se besitzt eine App getreu dem *Least-Privilege*-Prinzip keinerlei Berechtigungen, d. h. alle

Service Features sind initial deaktiviert. Daher sollte der Nutzer direkt bei der Installation die *Service Features* aktivieren, die er verwenden möchte. Welche Möglichkeiten ihm diesbezüglich von der *PMP* geboten werden, ist in Abschnitt 4.2.2 beschrieben und in Abbildung 4.7 gezeigt.

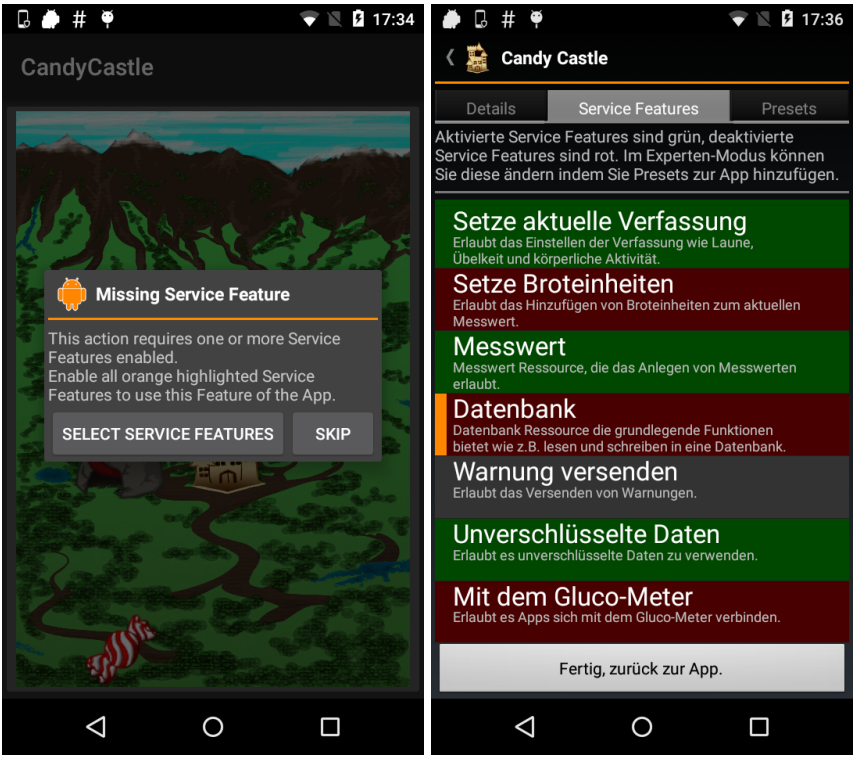
4.2.1 Feedback für den Nutzer

Damit ein Berechtigungssystem erfolgreich sein kann, d. h. dass der Nutzer damit umgehen kann und seine Interessen bezüglich Datenschutzfragen wahren kann, ist es entscheidend, wie er bei der Vergabe der Berechtigungen miteinbezogen wird [FEF+12]. Daher ist die *PMP* darauf ausgelegt, den Nutzer stets umfassend über seine Möglichkeiten bei der Vergabe von Berechtigungen zu informieren und ihm auch mögliche Konsequenzen seiner Einstellungen vor Augen zu führen.

Da die *PMP* dem *Least-Privilege*-Prinzip folgt, kommt es häufig vor, dass einer App nötige Berechtigungen fehlen, bzw. *Service Features* deaktiviert sind, die der Nutzer zumindest zeitweise verwenden möchte. Versucht die App ein *Service Feature* zu starten, das vom Nutzer deaktiviert wurde oder das aufgrund von *Privacy-Settings*-Einstellungen nicht ausgeführt werden kann, bemerkt dies die *PMP* und pausiert die App zwischenzeitlich. Der Nutzer bekommt von der *PMP* ein Informationsfenster angezeigt, dass er die *Privacy Settings* konfigurieren muss, damit die App das *Service Feature* ausführen kann (siehe Abbildung 4.6a).

Ist er nicht bereit die *Privacy-Settings* zu ändern, d. h. der App mehr oder genauere Daten zu geben, so setzt die *PMP* die App fort und überspringt das *Service Feature*. Will er aber das *Service Feature* nutzen, so springt die *PMP* zum Konfigurationsdialog, mit dem die *Privacy Rules* bearbeitet werden können. Zusätzlich blendet die *PMP* in diesem Fall eine orangene Markierung ein, die aufzeigt, welches *Service Feature* für den geplanten Ablauf der App aktiviert werden muss. Dies ist in Abbildung 4.6b bei der Datenbank-*Resource Group* zu sehen. Eine detaillierte Beschreibung dieses Dialogs ist in Abschnitt 4.2.2 gegeben.

Wenn der Nutzer weitere Informationen benötigt, welche Daten er bei der Aktivierung des *Service Features* preisgibt, so kann er sich diese von der *PMP* anzeigen lassen. Einerseits klärt die *PMP* ihn darüber auf, welche *Resources*

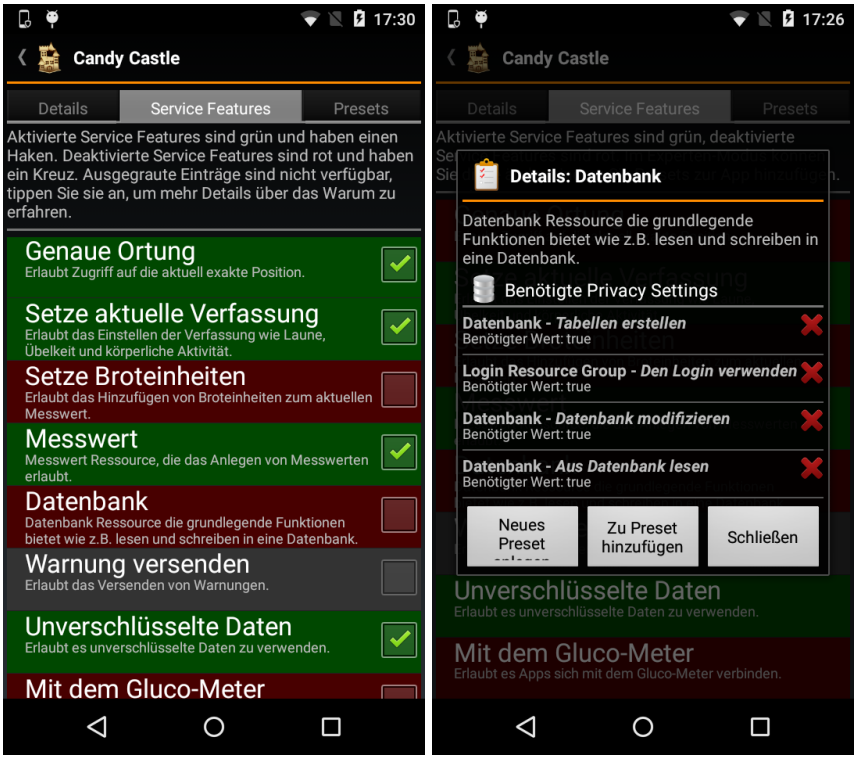


(a) Information über fehlende Berechtigungen

(b) Feedback zu deaktivierten *Service Features*

Abbildung 4.6: Feedbackmöglichkeiten der *PMP*

involviert sind (siehe Abbildung 4.7b). Andererseits kann er sich auch weitere Details zu jeder einzelnen *Resource Group* anzeigen lassen. Zu diesen Details zählen u. a. welche Daten exakt an die App weitergegeben werden, welche *Privacy Settings* für diese *Resource Group* existieren und welche Konfigurationsmöglichkeiten er dadurch besitzt (siehe Abbildung 4.9b). Über die *Privacy Settings* ist es ihm stets auch möglich, verfälschte Daten an die App weiterzuleiten. Dadurch kann das *Service Feature* ausgeführt werden, ohne private Daten mit der App teilen zu müssen. Die Konfigurationsmöglichkeiten der *Privacy Settings* sind in Abschnitt 4.2.3 beschrieben.



(a) Einfacher Modus (b) Expertenmodus

Abbildung 4.7: Änderung der Berechtigungen

4.2.2 Regeländerungen zur Laufzeit

Will der Nutzer neue *Privacy Rules* anlegen bzw. bestehende *Privacy Rules* ändern (und damit Einfluss auf die Berechtigungen seiner Apps nehmen), so bietet die *PMP* ihm grundsätzlich zwei unterschiedliche Möglichkeiten hierfür an. Im einfachen Modus kann er nur *Service Features* aktivieren bzw. deaktivieren. Wie im Schema für die *Privacy Policy* (Abbildung 4.3) zu sehen ist, sind mit jedem *Service Feature* stets die dafür erforderlichen *Privacy Settings* verknüpft. Im einfachen Modus muss sich der Nutzer darum aber nicht kümmern. Aktiviert er ein *Service Feature*, so setzt die *PMP* automatisch alle *Privacy Settings* entsprechend der Anforderungen des *Service Features*.

Abbildung 4.7a zeigt den Konfigurationsbildschirm des einfachen Modus. Grün hinterlegte *Service Features* sind aktiviert und rot hinterlegte *Service Features* sind deaktiviert. Durch einen Klick auf ein *Service Feature* wird dieses aktiviert respektive deaktiviert. Grau hinterlegte *Service Features* lassen sich aktuell nicht aktivieren, da diese auf *Resource Groups* zugreifen wollen, die auf dem System nicht verfügbar sind. Sie können erst aktiviert werden, wenn alle benötigten *Resource Groups* installiert wurden. In Abschnitt 4.2.5 wird beschrieben, wie neue *Resource Groups* dem System hinzugefügt werden können.

Im Expertenmodus hat der Nutzer wesentlich mehr Konfigurationsmöglichkeiten. Grundsätzlich werden ihm ebenfalls die *Service Features* der App im Grün-Rot-Grau-Schema angezeigt. Bei einem Klick darauf wird ein *Service Feature* allerdings nicht automatisch aktiviert, sondern es werden alle dafür benötigten *Privacy-Settings*-Einstellungen in einem weiteren Fenster angezeigt (siehe Abbildung 4.7b). Dieses Fenster informiert den Nutzer, welche *Privacy Settings* mit dem *Service Feature* verknüpft sind und inwiefern seine aktuellen *Privacy Rules* die Anforderungen des *Service Features* bereits erfüllen. Der Nutzer kann anschließend entweder alle *Privacy Settings* gemäß den Anforderungen konfigurieren oder einzelne *Privacy Settings* manuell setzen (siehe Abschnitt 4.2.3). Diese *Privacy Rules* kann er dann in einem neuen *Preset* speichern oder einem bestehenden hinzufügen, wenn er sie später exportieren und mit anderen Nutzern oder Geräten teilen möchte. Details zu den *Presets* sind in Abschnitt 4.2.7 beschrieben.

Die *PMP* gestattet es dem Nutzer jederzeit auf diesen Konfigurationsbildschirm zu wechseln und die *Privacy Policy* anzupassen. Die Änderungen werden nach der Bearbeitung direkt in die interne Datenbank der *PMP* übernommen. Bei jeder Ausführung eines *Service Features* und jedem Zugriff auf eine *Resource* prüft die *PMP* ob dies der *Privacy Policy* aus dieser Datenbank entspricht. Somit werden Regeländerungen unmittelbar angewandt.

Im Gegensatz zu Berechtigungssystemen, die nur zur Installationszeit einer App eine Konfiguration der Berechtigungen ermöglichen, wie es bei den Android-*Permissions* der Fall ist[10], ist dieses Vorgehen wesentlich flexibler. So

[10] Die *Runtime Permissions* von Android 6.0 gestatten für einige *Permissions* eine Änderung zur Laufzeit, jedoch nicht so feingranular und angepasst auf den Bedarfsfall, wie es mit den *Privacy Rules* möglich ist.

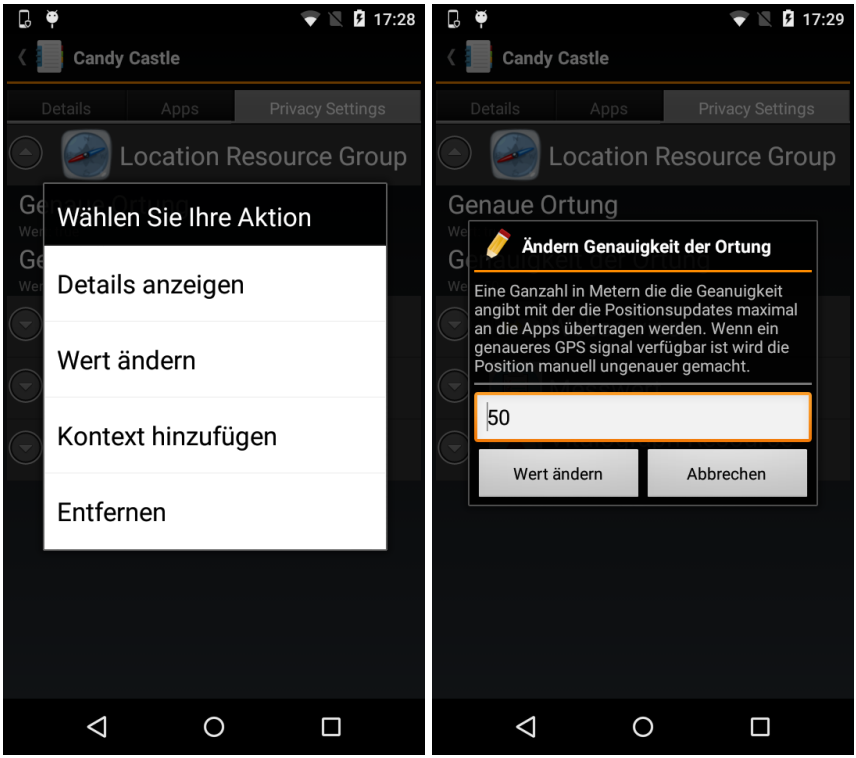
kann der Nutzer beispielsweise auch nur kurzzeitig (z. B. für eine einmalige Durchführung eines *Service Features*) die Rechte einer App ausweiten. Dadurch kann die Anzahl der vergebenen Berechtigungen und damit das Gefahrenpotenzial für die persönlichen Daten, das von der App ausgeht, minimalisiert werden.

4.2.3 Anonymisierung von Daten

Jede *Resource* besitzt mit der *Mock*-Implementierung und der *Cloak*-Implementierung zwei Implementierungsvarianten, die die Daten vollständig randomisieren (siehe Abschnitt 4.1.1). Zusätzlich zu den im *AIS* definierten *Privacy Settings* legt die *PMP* aus diesem Grund für jede *Resource* automatisch ein *Privacy Setting* namens *Modus* an. *Service Features* können für dieses *Privacy Setting* keine Vorgaben machen oder abfragen, welchen Wert der Nutzer für das *Modus-Privacy-Setting* festgelegt hat. D. h. der Nutzer kann frei darüber entscheiden, ob er randomisierte Daten verwenden möchte oder nicht, ohne dass dadurch *Service Features* deaktiviert werden. Wie die Daten randomisiert werden, hängt von der jeweiligen *Resource* ab; die *PMP* sorgt lediglich dafür, dass die richtige Implementierung aufgerufen wird.

Um diese Konfiguration durchführen zu können, muss der Nutzer lediglich die betreffende *Resource Group* auswählen und anschließend die Werte aller *Privacy Settings* dieser *Resource Group* ändern (siehe Abbildung 4.8a). Für das *Modus-Privacy-Setting* stehen ihm darauf die drei zulässigen Implementierungsarten zur Auswahl. Bei der *Cloak*-Implementierung ist es einer App nicht möglich, anhand der Daten festzustellen, dass diese verfälscht wurden. Daher ist diese vor allem dann sinnvoll, sollte eine App ihre Funktionalität reduzieren, wenn randomisierte Daten verwendet werden. Da dies in der Regel allerdings wesentlich rechenintensiver ist, empfiehlt es sich zunächst auf die *Mock*-Implementierung zurückzugreifen.

Zusätzlich zu der vollständigen Randomisierung der *Mock*-Implementierung respektive der *Cloak*-Implementierung, können mit mehrwertigen *Privacy-Settings*-Typen wie beispielsweise dem `EnumPrivacySetting` oder auch dem `IntegerPrivacySetting` die Daten einer *Resource* verfremdet werden, um dadurch die Privatsphäre des Nutzers zu wahren. Dies hängt allerdings von der *Resource* ab, ob diese solche *Privacy Settings* unterstützt. In Abbildung 4.8b ist



(a) Informationsfenster zu Anonymisierungsmöglichkeiten (b) Konfiguration der Anonymisierung

Abbildung 4.8: Anonymisierungsmöglichkeiten der *PMP*

dies am Beispiel einer *Resource Group* für Standortdaten zu sehen. Diese bietet ein *Privacy Setting* an, mit dem die Genauigkeit der Standortdaten reduziert werden kann. Der Nutzer kann in einem Textfeld einen positiven ganzzahligen Wert für dieses *Privacy Setting* vom `IntegerPrivacySetting`-Typ angeben. Bei *Privacy Settings* vom `EnumPrivacySetting`-Typ wird hingegen ein Auswahlmenü mit den zulässigen Werten angezeigt. Die Verfremdung findet anschließend in der jeweiligen *Resource* statt, da die hierfür nötigen Techniken stets von der Art der Daten, mit denen die *Resource* umgeht, abhängt.

4.2.4 Absturzsicherheit

Die heutigen mobilen Plattformen weisen Apps spätestens bei der Installation Berechtigungen zu und erlauben es nicht diese den Apps später wieder zu entziehen. Sollte der Nutzer dies dennoch versuchen (z. B. durch den Einsatz von Drittsoftware), so führt dies zu einem Absturz der App, sobald diese auf Daten oder Funktionen zugreift, ohne die dafür nötigen Berechtigungen zu besitzen (siehe Kapitel 2). Viele erweiterte Berechtigungssysteme für diese Plattformen stellen dem Nutzer allerdings nur Funktionen zur Verfügung, mit denen er die Berechtigungsdatenbank der Plattform dahingehend manipuliert, dass Apps Berechtigungen entzogen werden. Beispiele hierfür stellen u. a. *Cyanogen OS Privacy Guard*[11] oder auch Googles *App Ops*[12] dar, das mit Android 4.3 kurzzeitig eingeführt wurde. Beide Systeme wurden aber nach kurzer Zeit aufgrund von zu vielen Sicherheits-Exceptions wieder eingestellt. Erst nach weitgreifenden Überarbeitungen wurden die Systeme erneut veröffentlicht. Das *Cyanogen OS Privacy Guard* löst das Problem der Abstürze von Apps dadurch, dass leere Daten an eine App weitergegeben werden, wenn diese keine Zugriffsberechtigungen für die Daten besitzt. Google ließ das Konzept der *App Ops* in die *Runtime Permissions* einfließen; hier muss der App-Entwickler mit fehlenden Berechtigungen umgehen können, d. h. er muss die Sicherheits-Exception behandeln.

Die *PMP* bzw. das *PPM* berücksichtigen das Problem der Absturzsicherheit direkt von Beginn an. Einerseits definieren Apps *Service Features*, die individuell aktiviert oder deaktiviert werden können. Fehlt einer App eine Berechtigung, so kann das entsprechende *Service Feature* einfach übersprungen werden, ohne dass dadurch die App an sich betroffen ist. Darüber hinaus können durch die *Mock*-Implementierung respektive durch die *Cloak*-Implementierung stets Daten an eine App weitergeleitet werden, ohne dass dieser dadurch private Daten preisgegeben werden. Besonders bei der *Cloak*-Implementierung handelt es sich um realistische Daten. Dadurch können Apps in der Regel damit besser umgehen, als wenn leere Daten übertragen werden. Beispielsweise würde eine Standort-*Resource-Group* Werte zurückliefern, die echte Standortdaten darstellen. Würde hingegen ein `null`-Objekt zurückgegeben werden, so muss

[11] siehe http://www.cyanogenmod.org/
[12] siehe Amadeo [Ama13]

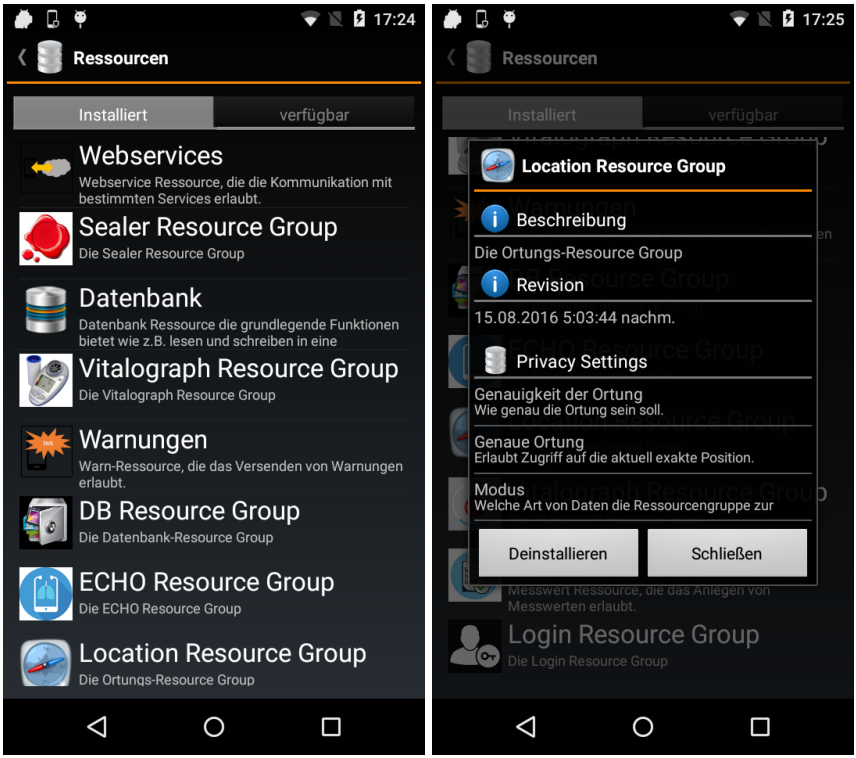
die App damit umgehen können; ansonsten führt dies ebenfalls zu einem Absturz.

Somit verbindet die *PMP* nicht nur die beiden Strategien von *Cyanogen OS Privacy Guard* und Googles *App Ops*, sondern optimiert diese weiter. Dadurch garantiert die *PMP* Absturzsicherheit und ermöglicht eine problemlose Ausführung der App mit bestmöglicher und auf den Nutzer zugeschnittener Servicequalität auch bei eingeschränkten Berechtigungen.

4.2.5 Erweiterbarkeit

Das Konzept der *PMP* sieht es vor, dass sich diese auf technische Neuerungen in zukünftigen Generationen von Smart Devices einstellen kann. Da sich die Anzahl und Art der darin verbauten Sensoren stetig weiterentwickelt, werden auf diesen Geräten auch immer mehr und immer unterschiedlichere Daten gespeichert [Pin15]. Um diese Daten schützen zu können, muss sich auch ein Datenschutzsystem weiterentwickeln können. Damit der Nutzer einerseits nicht dauerhaft Updates für die *PMP* einspielen und andererseits keine Erweiterungen für die Unterstützung von Sensoren installieren muss, die in seinem Gerät gar nicht verbaut sind, lassen sich der *PMP* dynamisch *Resource Groups* hinzufügen.

Über das Hauptmenü der *PMP* kann der Nutzer sich alle *Resource Groups* anzeigen lassen (siehe Abbildung 4.9a). Weitere verfügbare *Resource Groups* können von hier aus direkt über einen Server der *PMP* bezogen werden. Hier verfolgt die *PMP* das gleiche Prinzip wie Apple, in dem es nur eine zulässige Bezugsquelle für *Resource Groups* gibt. Dies hat den Hintergrund, dass es sich hierbei um sicherheitskritische Komponenten handelt und das *PPM* darauf beruht, dass die *Resource Groups* vertrauenswürdig sind. Daher müssen sie besonders überprüft werden, bevor sie dem Nutzer zur Verfügung gestellt werden. Da die Anzahl der *Resource Groups* allerdings verglichen mit der Anzahl von neu entwickelten Apps gering ist, hält sich der Aufwand für die Kontrolle der *Resource Groups* in Grenzen. Auch kann diese Kontrolle im Gegensatz zu Apples Überprüfungen nicht als Zensurmechanismus missbraucht werden, da die *Resource Groups* für sich genommen dem Nutzer keine Funktionalität anbieten – erst wenn eine App eine *Resource Group* nutzt, profitiert der Nutzer von dieser.



(a) Auflistung der installierten *Resource Groups*

(b) Informationsfenster zu den *Resource Groups*

Abbildung 4.9: Erweiterungsmöglichkeiten der *PMP* durch die *Resource Groups*

Am besten lässt sich der Vorteil dieses um *Resource Groups* erweiterbaren Modells anhand von Sensoren demonstrieren, die mittels Bluetooth mit dem Smart Device verbunden sind. Gerade im *mHealth*-Sektor kommen viele Messgeräte mit Bluetooth-Schnittstellen zum Einsatz [CEF+12]. Dabei stehen App-Entwickler vor zwei großen Herausforderungen, da sie einerseits die Sicherheit der Daten gewähren und andererseits für jedes Gerät ein eigenes, teilweise sogar proprietäres Kommunikationsprotokoll einhalten müssen [BDL14]. Android stellt für solche Geräte nur eine einzige *Permission* für den Zugriff auf Geräte, die über Bluetooth verbunden sind, zur Verfügung. D. h. jede App,

die die Lautstärke von Bluetooth-Kopfhörer regeln kann, darf beispielsweise ebenfalls Blutzuckermessgeräte auslesen oder moderne Insulinpumpen steuern, ohne dass der Nutzer dies mitbekommt! In der *PMP* kann hingegen für jedes Bluetooth-Gerät eine eigene *Resource Group* angelegt werden und der Nutzer kann anschließend, vergleichbar mit der Installation eines Treibers für sein Gerät, die für ihn passende *Resource Group* installieren. Die *PMP* setzt ihn darüber in Kenntnis, welche *Resource Groups* für ihn in Frage kommen und welche von Apps, die bei ihm installiert sind, angefordert werden. Der Vorteil der separaten *Resource Groups* besteht nicht nur in der feingranularen Berechtigungsvergabe, sondern auch darin, dass jede *Resource Group* eigene *Privacy Settings* definieren kann, die für den jeweiligen Gerätetyp passend sind. So reicht bei einem Kopfhörer sicherlich ein einziges boolesches *Privacy Setting* aus, während ein medizinisches Messgerät weitere *Privacy Settings* anbieten kann, um die Art und Genauigkeit der Daten, die mit einer App geteilt werden, besser zu reglementieren. Die *PMP* informiert den Nutzer bei jeder *Resource Group* über die darin definierten *Privacy Settings* und deren zulässigen Wertebereiche. Damit ist er in der Lage, die *Privacy Rules* bestmöglich nach seinen Vorstellungen zu erstellen (siehe Abbildung 4.9b).

Für den App-Entwickler besteht der Vorteil darin, dass die *Resources* sich um die Kommunikation mit den externen Geräten kümmern und er sich um Kommunikationsprotokolle keine Gedanken machen muss. Da gleichartige *Resources* in einer *Resource Group* zusammengefasst werden können, muss er lediglich die öffentliche Schnittstelle dieser *Resource Group* ansprechen, um mit dem Gerät zu kommunizieren. Ein Beispiel hierfür sind unterschiedliche Bluetooth-Blutzuckermessgeräte (mit jeweils individuellen Kommunikationsprotokollen), für die jeweils eine *Resource* existiert, die alle in Blutzucker-*Resource-Group* gruppiert sind. Der App-Entwickler muss sich nicht darum kümmern, welche *Resource*, d. h. welches Gerät, beim Nutzer zum Einsatz kommt, sondern spricht direkt die komplette *Resource-Group* an. Die *PMP* stellt bei der Einbindung der *Resource-Group* sicher, welche *Resource* verwendet werden soll.

Aber nicht nur die Interoperabilität von Apps und die Datensicherheit wird durch die Erweiterbarkeit der *PMP* gefördert, sondern auch die Qualität von Apps kann dadurch gefördert werden. In einer *Resource* lassen sich häufig verwendete Funktionalitäten auslagern. Dies muss nicht zwangsweise mit

einem Sensor in Verbindung stehen. Jede Art von Code, der Daten verarbeitet, kann in einer *Resource* hinterlegt werden und steht somit App-Entwicklern zur Verfügung. Dadurch können Fehlerquellen reduziert werden, da dieser redundante Code nur an einer zentralen Stelle gepflegt und optimiert werden muss und nicht in jeder App aufs Neue.

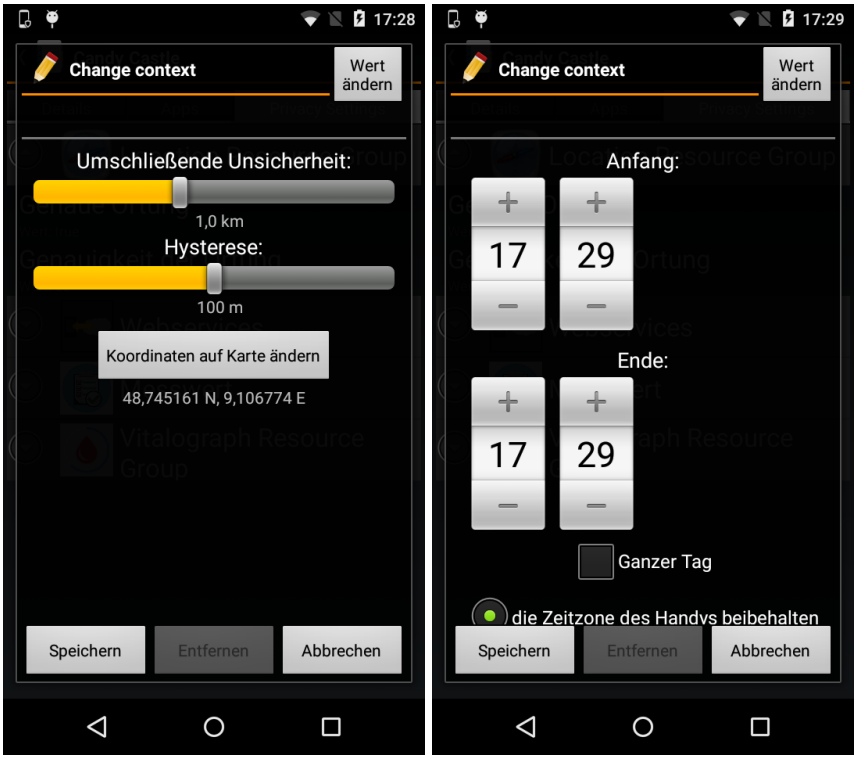
Darüber hinaus können *Resources* auch für einen ausgewählten Anwendungsfall spezialisiert werden. So wäre eine *Resource* denkbar, die eine Internetverbindung aufbaut und Daten darüber versendet. Allerdings kann die Datensenke auf eine oder mehrere ausgewählte Adressen eingeschränkt werden. Bei den heutigen mobilen Plattformen wird gerade bemängelt, dass Apps Daten an jede beliebige Senke schicken können, sobald sie eine Berechtigung für den Internetzugriff haben, und der Nutzer keine Möglichkeit besitzt nachzuvollziehen, wer somit seine Daten erhält [BKvOS10]. Mit einer spezialisierten Internet-*Resource* kann die *PMP* hingegen sicherstellen, dass Daten nur an die im Vorfeld vereinbarten Senken weitergeleitet werden.

4.2.6 Kontextsensitivität

Häufig hängen die Berechtigungen einer App in der Realität von einer Situation ab, in der der Nutzer sie anfordert. Dadurch werden die Berechtigungen flexibler und können feingranularer vergeben werden [HSB+05]. Ein solches kontextbasiertes Berechtigungsmodell lässt sich auch auf mobile Plattformen übertragen. Hierbei bestimmt der Kontext, der durch die Sensoren des Smart Devices erfasst werden kann, welche Berechtigungen eine App besitzt [BGF+10].

Auch die *PMP* berücksichtigt daher in ihrem *PPM* den Kontext. Prinzipiell kann der Kontext durch jede Art von Daten, die von *Resources* bereitgestellt werden, beschrieben werden, z. B. Standortdaten oder die Zeit bzw. das Datum. Bei diesen beiden Datenarten handelt es sich, neben den beteiligten Instanzen und der durchgeführten Aktivität, um die relevantesten Daten zur Beschreibung des Kontextes [DA99; Dey01]. Die Instanzen (Apps und *Resource Groups*) und die durchgeführten Aktivitäten (*Service Features*) werden im *PPM* ohnehin berücksichtigt.

Zur Beschreibung des Ortskontextes stellt die *PMP* dem Nutzer eine einfache Eingabemaske zur Verfügung (siehe Abbildung 4.10a). Damit kann er zunächst



(a) Definition des Ortskontextes (b) Definition des Zeitkontextes

Abbildung 4.10: Definitionsmöglichkeiten des Kontextes

auf einer Karte einen Ort auswählen und mittels Schiebereglern feinjustieren, in welchem Bereich um diesen Ort herum eine *Privacy Rule* gelten soll. Auch zur Beschreibung des Zeitkontextes besitzt die *PMP* eine Eingabemaske (siehe Abbildung 4.10b). Über Datums- und Zeitfelder kann der Nutzer definieren, zu welchen Zeiten eine *Privacy Rule* gelten soll.

Auf diese Weise kann der Nutzer beispielsweise seinen Arbeitsplatz als Ortskontext auswählen und seine Arbeitszeit als Zeitkontext und damit eine *Privacy Rule* annotieren. Diese Regel würde von der *PMP* nur angewendet werden, wenn sich der Nutzer zur Arbeitszeit an seinem Arbeitsplatz aufhält. Somit kann der Nutzer *Privacy Rules* der *Privacy Policy* hinzufügen, die konform

zu den Richtlinien seines Arbeitgebers sind, aber ausschließlich dann angewandt werden, während er arbeitet. Im Rahmen einer Diplomarbeit von Panos [Pan13] wurde untersucht, wie dadurch die *PMP* zum Enabler für Bring Your Own Device (BYOD)[13] wird.

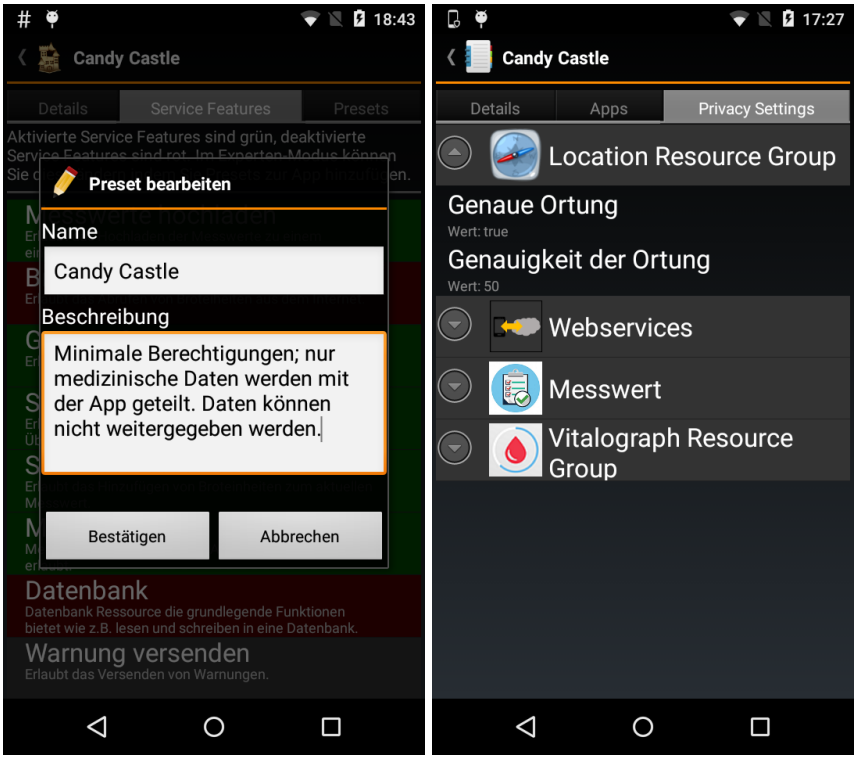
4.2.7 Presets

Zur Erstellung der *Privacy Policy* erfährt der Nutzer durch die *PMP* sehr viel Unterstützung in Form von GUIs und ausführlichem Feedback. Allerdings kann die richtige Konfiguration der *Privacy Rules* aufgrund der feingranularen Einstellungsmöglichkeiten zeitaufwendig sein. Der einfache Modus verkürzt die Zeit für die Konfiguration zwar, jedoch führt er hinsichtlich der Berechtigungen nicht immer zu einem optimalen Ergebnis, da darin die *Privacy Settings* nicht manuell angepasst werden können. Doch selbst im einfachen Modus verursacht die Konfiguration vieler Apps einen hohen Aufwand. Besonders Laien könnten sich daher bei der Erstellung der *Privacy Policy* überfordert fühlen, wenn sie die Berechtigungen sehr vieler Apps über die *PMP* regulieren wollen.

Aus diesem Grund unterstützt die *PMP* sogenannte *Presets*. Ein *Preset* beschreibt eine Teilmenge der *Privacy Rules* aus der *Privacy Policy*. Diese *Presets* können anschließend über die *PMP* exportiert und verteilt bzw. auch importiert werden. Jedes *Preset* wird über einen Namen und einen Beschreibungstext charakterisiert (siehe Abbildung 4.11a). Der Nutzer kann sich die im *Preset* hinterlegten *Service Features* sowie die Konfiguration für die damit verbundenen *Privacy Settings* anzeigen lassen und auch verändern (siehe Abbildung 4.11b).

Einerseits kann sich ein Nutzer so auf einem Gerät eine für ihn optimale *Privacy Policy* zusammenstellen und diese mittels *Presets* auf alle seine restlichen Geräte übertragen. Andererseits kann er *Presets* als Entscheidungshilfe, wie eine sinnvolle *Privacy Policy* aussehen könnte, anderen Nutzern zur Verfügung stellen. Importieren diese Nutzer fremde *Presets*, so prüft die *PMP* automatisch, welche der darin enthaltenen *Privacy Rules* für sie relevant sind. D. h. es werden nur *Privacy Rules* importiert, die Apps betreffen, die ein Nutzer bei sich installiert hat. Vertrauenswürdige Instanzen könnten daher mit den *Presets* unerfahrenen

[13] BYOD bedeutet, dass Arbeitnehmer ihre private Hardware, wie beispielsweise Smartphones, zur Erfüllung ihrer Aufgaben nutzen und dabei auf die Infrastruktur des Arbeitgebers zugreifen dürfen.



(a) Erstellungsdialog für ein *Preset* (b) Details zu einem *Preset*

Abbildung 4.11: Unterstützung der *Presets*

Nutzern eine Art Basiskonfiguration als Sicherheitsempfehlung anbieten. Das Bundesamt für Sicherheit in der Informationstechnik – BSI [BSI16] stellt für Browser bereits eine solche Richtlinie bereit.

Importiert ein Nutzer mehrere *Presets* aus unterschiedlichen Quellen, so kann es dabei zu Konflikten kommen. Ein Konflikt bedeutet in diesem Kontext, dass in diesen *Presets Privacy Rules* für die gleichen Apps hinterlegt sind, die sich widersprechen. Die *PMP* versucht anschließend den Konflikt aufzulösen. Stehen die *Privacy Rules* in keinen Widerspruch zueinander, so werden alle *Presets* automatisch additiv kombiniert. Lässt sich der Konflikt nicht auflösen, so werden die betroffenen *Privacy Rules* hervorgehoben, und der Nutzer kann

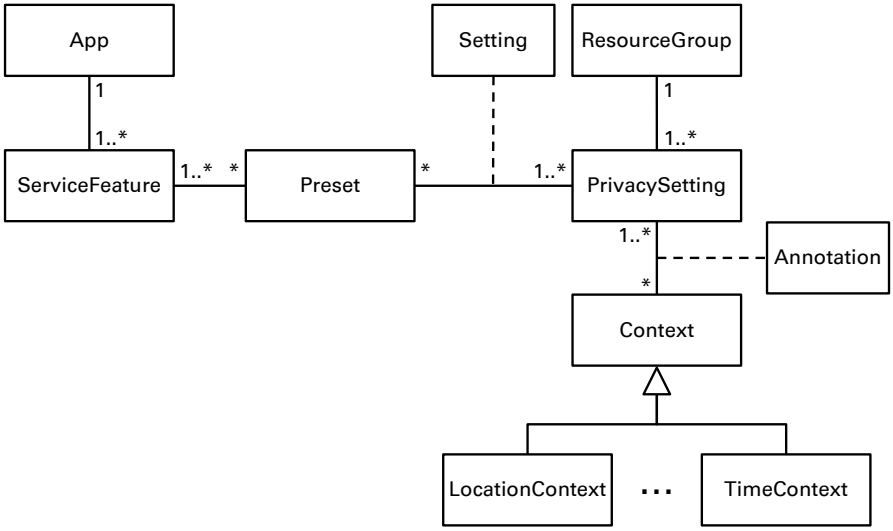


Abbildung 4.12: Modell des Datenaustauschformats für *Presets*

entscheiden, welche der Regeln er verwerfen möchte, um den Konflikt so manuell aufzulösen.

Das den *Presets* zugrundeliegende Klassenmodell ist in Abbildung 4.12 dargestellt. Die *PMP* verwendet dieses Datenformat anstelle von beispielsweise XML oder JavaScript Object Notation (JSON), obwohl diese wesentlich leichtgewichtiger sind und daher gerade im Bereich mobiler Plattformen häufig zum Einsatz kommen. Der Vorteil serialisierter Java-Objekte ist, dass die gespeicherten Daten darin nicht in Klartext vorliegen und somit von außen nicht verändert werden können. Dies könnte auch für Formate, in denen die Daten im Klartext vorliegen, dadurch erreicht werden, indem Kryptographiealgorithmen oder digitale Signaturen zum Einsatz kommen. Dies würde aber bei der Erstellung sowie bei dem Im- und Export von *Presets* einen Rechen-Overhead erzeugen. Daher baut die *PMP* auf das schwergewichtigere Datenformat der serialisierten Java-Objekte.

4.3 Implementierungsstrategien

Im Folgenden werden Implementierungsdetails des *PPMs* sowie unterschiedliche Implementierungsstrategien für ein Datenschutzsystem wie die *PMP* diskutiert. Die Implementierung der *PMP* die dieser Arbeit zugrunde liegt, wurde für Android angefertigt, weshalb die Implementierungsdetails stark mit dieser mobilen Plattform verbunden sind. Die diskutierten Konzepte sind allerdings plattformübergreifend und lassen sich auf jede andere App-Plattform oder Betriebssystem anwenden.

Für die Inter-Process Communication (IPC) kommt in der *PMP* das auf *OpenBinder*[14] basierende *Binder*-Konzept von Android zum Einsatz. Mit einem *Binder* kann ein Prozess Funktionen oder Daten anderen Prozessen zur Verfügung stellen. Hierfür muss der *Binder* eine sogenannte *IBinder*-Schnittstelle definieren und implementieren. Das in Android implementierte *Binder*-Framework nimmt dem Entwickler bei der Kommunikation zwischen den beiden Prozessen viel Verantwortung ab. So erzeugt es anhand der *IBinder*-Schnittstelle in AIDL einen *Stub* genannten Platzhalter für die eigentliche Implementierung der Schnittstelle. Um eine Schnittstellenimplementierung anzugeben, muss der Entwickler nur diese *Stub*-Klasse erweitern. Die Bindung dieser konkreten Implementierung an die *Stub*-Klasse und damit an die *Binder*-Schnittstelle erfolgt dynamisch. Zu jedem *Stub* stellt das *Binder*-Framework einen *Proxy* zur Verfügung, über den andere Prozesse auf die in der Schnittstelle definierten Funktionen zugreifen können. Hierfür benötigt der anfragende Prozess ein *Binder Token*, das den *Binder* eindeutig identifiziert. Die Methodenaufrufe an den externen Prozess, d. h. den Prozess, in dem die Methoden implementiert sind, können dabei durchgeführt werden, als ob es sich um lokale Methoden des anfragenden Prozesses handelt, da durch die *Stubs* sichergestellt werden kann, dass mindestens eine Pseudoimplementierung der Methoden vorliegt. Die Anfrage kann sowohl synchron als auch asynchron erfolgen[15] [Hac07; And16c].

Abbildung 4.13 stellt das Implementierungsmodell der *PMP* dar. Zunächst stellt ein *Service Feature* über die Application Programming Interface (API)

[14] siehe http://www.angryredplanet.com/ hackbod/openbinder/docs/html/

[15] Das Binder-Framework ist in der Lage den anfragenden Prozess zu pausieren und wieder aufzuwecken, wenn die Daten des externen Prozesses vorliegen.

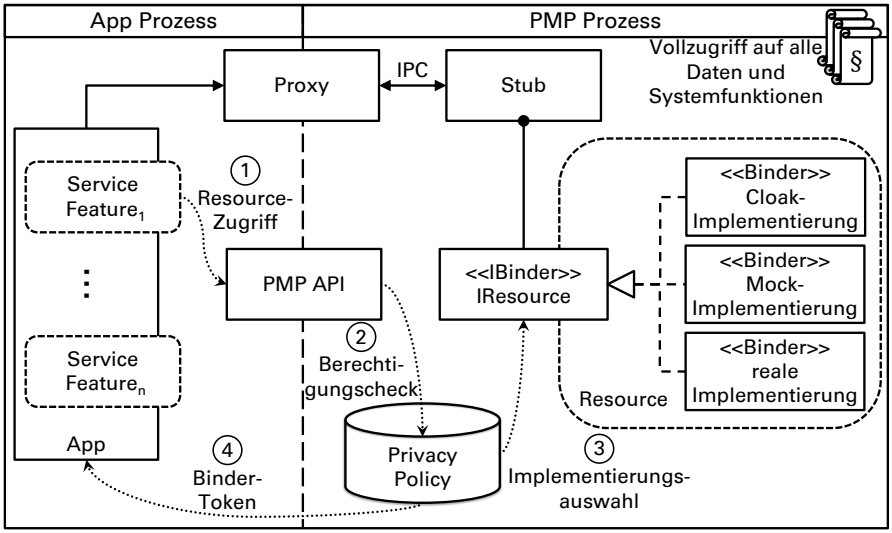


Abbildung 4.13: Vereinfachtes Implementierungsmodell der *PMP*

der *PMP* eine Anfrage, wenn von einer App Zugriff auf eine *Resource* benötigt wird (1). Die *PMP* prüft daraufhin in der *Privacy Policy* nach, ob die Anfrage gemäß der definierten *Privacy Rules* durchgeführt werden darf und welche Einstellungen bezüglich der Anonymisierungstechnik dabei gelten (2). Falls der Zugriff zulässig ist, wird abhängig von den *Privacy Rules* eine Implementierungsvariante der *Resource* ausgewählt (die *Cloak*-, *Mock*- oder die reale Implementierung) (3). Diese Implementierung wird anschließend als *Binder* an die *IBinder*-Schnittstelle der *Resource* durch die *PMP* gebunden. Das *Binder*-Token wird anschließend von der *PMP* an die anfragende App zurückgegeben (4).

Der eigentliche Zugriff auf die *Resource* erfolgt über den vom *Binder*-Framework zur Verfügung gestellten *Proxy*, der an den *Stub* der *Resource* gebunden ist. Ein direkter Zugriff auf eine *Resource* oder deren *Proxy* ohne das zugehörige *Binder*-Token ist nicht möglich. Daher ist sichergestellt, dass jede Anfrage über die *PMP* laufen muss und diese stets überprüfen kann, ob der Zugriff auf die *Resource* gemäß der *Privacy Policy* zulässig ist.

Für die Kommunikation zwischen der *PMP* und den daran angeschlossenen *Resources* sind keine IPC-Techniken nötig. Die *Resource Groups* müssen alle in speziellen Unterpaketen der *PMP* liegen und sie müssen die User Identifier (UID) sowie die Signatur der *PMP* besitzen. Unter diesen Bedingungen können unter Android Apps im gleichen Prozess und damit der gleichen Sandbox ausgeführt werden [And16g]. Dadurch kann die *PMP* direkt mit den *Resources* interagieren. Darüber hinaus ist auf diese Weise sichergestellt, dass *Resource Groups* aufgrund der benötigten Signatur nicht veröffentlicht werden können, ohne dass diese von den *PMP*-Entwicklern auf Schwachstellen oder Gefährdungen hin untersucht werden konnten.

Damit eine *Resource* auf die von ihr benötigten Daten zugreifen kann, muss diese zunächst die dafür erforderlichen Berechtigungen von der mobilen Plattform erhalten (z. B. im Fall von Android die jeweiligen *Permissions*). Die *PMP* löst dieses Problem dadurch, dass sie einen Vollzugriff auf alle Daten und Systemfunktionen besitzt[16]. Da die *Resources* im gleichen Prozess wie die *PMP* ausgeführt werden, können diese über die *PMP* ebenfalls auf die Daten zugreifen. Daher benötigen weder Apps noch *Resources* jedwede Berechtigungen der mobilen Plattform, sondern erhalten alle Daten über die *PMP*, sofern dies der *Privacy Policy* entspricht. Dadurch wird das Berechtigungssystem der mobilen Plattform obsolet und die *PMP* sorgt stattdessen für ein feingranulares und nutzerfreundliches Berechtigungsmanagement.

Um ein Datenschutzsystem wie die *PMP* in eine bestehende mobile Plattform einzubinden, gibt es mehrere Implementierungsstrategien, die in Abschnitt 4.3.1 bis Abschnitt 4.3.5 diskutiert werden. Abschnitt 4.3.6 evaluiert abschließend, welche Implementierungsvariante für die *PMP* am Besten geeignet ist. Dabei wird stets analysiert, wo bezüglich der Android-Systemarchitektur (siehe Abbildung 2.3) in der jeweiligen Implementierungsstrategie Anpassungen vorgenommen werden müssen und welche Auswirkungen dies auf die Funktionsweise und Sicherheit des Systems hat.

[16] Für einen echten Vollzugriff auf Systemdaten sind *Root*-Rechte erforderlich. Es reicht allerdings aus, wenn die *PMP* alle Berechtigungen besitzt, die Apps von Drittanbietern erhalten können. Dadurch ist sie in der Lage auf alle Daten und Funktionen zuzugreifen, die Apps ohne *Root*-Rechten zur Verfügung stehen, und kann diese über die *Resources* an andere Apps weitergeben.

4.3.1 Erweiterung der bestehenden Berechtigungsregeln

Mobile Plattformen führen mit neuen Versionen des Betriebssystems auch häufig Änderungen an den Berechtigungen durch; es werden bestehende Berechtigungen ersetzt respektive verfeinert oder es werden neue Berechtigungen eingeführt. So wurde beispielsweise von Android mit API-Level 19, d. h. mit Version 4.4, die `BLUETOOTH_PRIVILEGED`-*Permission* als Erweiterung zu der bestehenden `BLUETOOTH`-*Permission* eingeführt. Apps, die diese *Permission* haben, können sich dadurch jederzeit mit Bluetooth-Geräten verbinden, ohne dass dies zunächst durch den Nutzer eingeleitet werden muss; Apps mit der normalen `BLUETOOTH`-*Permission* dürfen dies nicht. Durch diese Verfeinerung und Aufspaltung der `BLUETOOTH`-*Permission* können App-Entwickler nun sehr viel feingranularer festlegen, welche Berechtigung am Besten zu der Anforderung ihrer App passt.

Die `USE_FINGERPRINT`-*Permission* ist hingegen ein Beispiel für eine *Permission*, die in Android mit API-Level 23, d. h. mit Version 6.0, vollkommen neu eingeführt wurde. Apps, die diese Berechtigung besitzen, dürfen auf Hardwarekomponenten zugreifen, mit denen Fingerabdrücke identifiziert werden können. Da diese Hardware allerdings erst mit neueren Gerätegenerationen eine weitere Verbreitung fand, bestand davor auch kein Bedarf für eine solche *Permission*.

Solche neuen *Permissions* werden in der Regel nicht direkt mit der Einführung aktiviert, d. h. Apps können diese *Permission* zwar bereits anfordern, aber der Zugriff auf Hardwarekomponenten zur Fingerabdruckerkennung ist innerhalb einer Übergangszeit auch ohne diese *Permission* möglich. Dadurch haben Drittanbieter eine Möglichkeit, auf Revisionen an den *Permissions* zu reagieren, und Updates für ihre Apps zu veröffentlichen, falls diese von den Änderungen betroffen sind.

Aus Nutzersicht ergibt sich zunächst keine Veränderung, da er in der Regel keinen Überblick darüber hat, welche *Permissions* existieren. Die mobile Plattform informiert ihn lediglich bei der Installation darüber, welche *Permissions* von einer App angefordert werden, bzw. bei einem Update, ob bei einer App neue oder andere *Permissions* hinzukamen.

Grundlegende Neuerungen, wie beispielsweise kontextsensitive Berechtigungen oder Berechtigungsregeln, die sich zur Laufzeit ändern lassen, können

dadurch allerdings nicht realisiert werden. Im Fall von Android erhält der Nutzer weiterhin ausschließlich bei der Installation einer App sehr generische Informationen darüber, welche Berechtigungen diese App anfordert, aber nicht, warum sie diese benötigt. Der Einfluss, den diese Implementierungsstrategie auf den Datenschutz hat, ist daher nur sehr gering.

Noch schwerwiegender ist die Tatsache, dass nach einem automatischen Systemupdate manche Apps nicht mehr funktionieren, wenn mit dem Update neue Berechtigungen aktiv wurden. Erst nachdem der Nutzer alle Apps, die auf seinem Gerät sind, ebenfalls aktualisiert hat und diesen dabei die neu benötigten Berechtigungen gibt, sind diese wieder lauffähig. Dies setzt allerdings voraus, dass der App-Entwickler seine Apps kontinuierlich an neue Versionen der mobilen Plattform und damit an deren Revisionen des Berechtigungssystems anpasst. Da Apps jedoch häufig von Entwicklern stammen, die diese nur als Hobby erstellen, werden viele Apps nicht oder nur sehr unzureichend aktualisiert. Aber auch wenn die Entwickler ihre Apps stets auf dem neusten Stand halten, kann es zu Komplikationen kommen. Werden mit einem App-Update neue *Permissions* verwendet, die auf der Android-Version, die beim Nutzer läuft, noch nicht eingeführt wurden, so ist die App ebenfalls nicht ausführbar. Da die Plattformanbieter allerdings mit der Einführung einer neuen Hardwaregeneration oft keine Plattform-Updates mehr für ältere Geräte herausgeben, kann durch eine regelmäßige Wartung einer App ebenfalls nicht garantiert werden, dass diese bei allen Nutzern lauffähig ist.

Neben diesen schwerwiegenden Nachteilen spricht vor allem das marginale Verbesserungspotenzial gegen diese Implementierungsstrategie. Für die Einführung eines umfassenden Datenschutzsystems wie die *PMP* ist sie gänzlich ungeeignet. Lediglich für die Anbieter mobiler Plattformen stellt dies eine Möglichkeit dar, um das vorhandene Berechtigungssystem sukzessive an Systemänderungen (z. B. durch die Ablösung veralteter APIs) anzupassen oder um auf technische Neuerungen (z. B. durch die Einführung neuer Hardware) zu reagieren.

4.3.2 Implementierung als App

Soll neben neuen und erweiterten Berechtigungen auch ein verbesserter Durchsetzungsmechanismus zum Einsatz kommen, der den Nutzer stärker bei der

Festlegung der Berechtigungen miteinbezieht, oder gar ein vollständig neues Modell für die Zugriffsregeln eingeführt werden, wie es bei der *PMP* der Fall ist, muss hierfür eine Laufzeitumgebung geschaffen werden. Eine Möglichkeit stellt die anwendungsbasierte Realisierung des Datenschutzsystems dar. D. h. das System wird als App ausgeliefert, die auf der mobilen Plattform installiert werden kann. Ein Datenschutzsystem, das dieser Implementierungsstrategie folgt, kann mit einer Sicherheitssoftware, wie man sie von Desktop PCs kennt (z. B. Firewalls oder Virenscanner) verglichen werden. Das Betriebssystem der mobilen Plattform wird durch diese zusätzliche Software nicht beeinflusst.

Die Sicherheitssoftware hingegen unterliegt den gleichen Restriktionen und Bestimmungen, die für alle Apps von Drittanbietern gelten. Dadurch sind die Möglichkeiten, die eine solche Sicherheitssoftware hat, um andere Apps zu überwachen, stark reglementiert. Auch kann sie nicht ohne weiteres die Zugriffe anderer Apps auf System- und Nutzerdaten einschränken oder gar vollständig unterbinden.

Für die *PMP* bedeutet dies zweierlei Dinge: Einerseits benötigt sie selbst vollen Zugriff auf alle zu schützenden Daten. D. h. unter Android müssen im Manifest der *PMP* `uses-permission`-Einträge für alle von Android definierten Berechtigungen vorhanden sein. Allerdings gibt es unter Android einige System-*Permissions*, die nur Apps bekommen können, die von Google signiert werden. Beispielsweise kann die `DUMP`-*Permission* nicht an Apps vergeben werden, die von Drittanbietern stammen. Damit wird sichergestellt, dass keine App den dynamischen Speicher auslesen und auf diese Weise an vertrauliche Information oder Systemdaten gelangen kann. Dies stellt aber für die *PMP* keine Einschränkung dar, da aus Sicht eines Datenschutzsystems für diese Art von Berechtigungen nichts zu unternehmen ist – wenn keine App die *Permission* erhalten kann, muss die *PMP* Zugriffe, die nur mit der *Permission* möglich sind, auch nicht reglementieren. Verfügt die *PMP* über alle *Permissions*, die Apps von Drittanbietern bekommen können, ist sie in der Lage auf alle verfügbaren System- und Nutzerdaten zuzugreifen und diese bei Bedarf ggf. gekürzt oder randomisiert, an andere Apps weiterzuleiten. Zusätzlich muss sichergestellt werden, dass die *PMP* direkt beim Systemstart geladen wird und weder durch Apps noch durch den Nutzer beendet werden kann. Ansonsten kann nicht garantiert werden, dass alle Zugriffsversuche durch andere Apps festgestellt

und reglementiert werden. Außerdem hängt die Funktionalität der restlichen Apps, die auf der mobilen Plattform laufen, unmittelbar mit der Verfügbarkeit der von der *PMP* bereitgestellten Dienste zusammen. Nur wenn eine App die *Resources* erreichen kann, hat sie Zugang zu den von ihr benötigten Daten.

Andererseits muss die *PMP* alle Apps, die auf der mobilen Plattform installiert sind, kennen. Ohne *Root*-Rechte ist dies allerdings nicht direkt realisierbar. Daher müssen alle Apps, die mit der *PMP* zusammenarbeiten, d. h. die ihre Daten von der *PMP* beziehen und deren Dienste nutzen wollen, sich bei der Installation bei der *PMP* registrieren und bei jedem App-Start bei der *PMP* anmelden. Hierfür stellt die *PMP* Registrierungsaktivitäten zur Verfügung, die der eigentlichen App vorgeschaltet werden. Im Rahmen dieser Aktivität wird auch überprüft, ob die *PMP* läuft und, sollte dies nicht der Fall sein, dass diese automatisch zusammen mit der App gestartet wird.

Will eine App auf durch die *PMP* geschützte Daten zugreifen, so kann sie dies nach der Anmeldung gemäß Abbildung 4.13 direkt bei der *PMP* beantragen und diese führt alle erforderlichen Berechtigungschecks durch. Ein Zugriff auf das Anwendungsframework von Android durch die App ist hierdurch nicht nötig, da alle Zugriffe über die *PMP* erfolgen. Dies bedeutet darüber hinaus, dass eine App keinerlei Android-*Permissions* mehr benötigt. Problematisch ist bei dieser Implementierungsstrategie allerdings, dass die *PMP* gleichberechtigt mit jeder anderen App von Drittanbietern ist, d. h. sie kann nicht ausschließen, dass Apps nicht dennoch über ihr Manifest *Permissions* anfordern und damit heimlich versuchen, direkt über das Anwendungsframework auf System- und Nutzerdaten zuzugreifen.

Abbildung 4.14 zeigt, welche Anpassungen bei dieser Implementierungsstrategie an der Android-Systemarchitektur erforderlich sind. An der eigentlichen mobilen Plattform finden keine Änderungen statt. Die *PMP* sowie alle *Resources* werden als Apps der Anwendungsschicht hinzugefügt. Der Vorteil für den Nutzer besteht darin, dass alle Apps der mobilen Plattform (nachfolgend als *System-Apps* bezeichnet) und alle Apps von Drittanbietern, die nicht für die Zusammenarbeit mit der *PMP* angepasst wurden (nachfolgend als *Legacy-Apps* bezeichnet), auch nach der Einführung der *PMP* problemlos funktionieren. Diese Apps greifen auf die von ihnen benötigten Daten weiterhin direkt über das Anwendungsframework zu, und nicht über die *PMP*. Gerade für unerfahrene

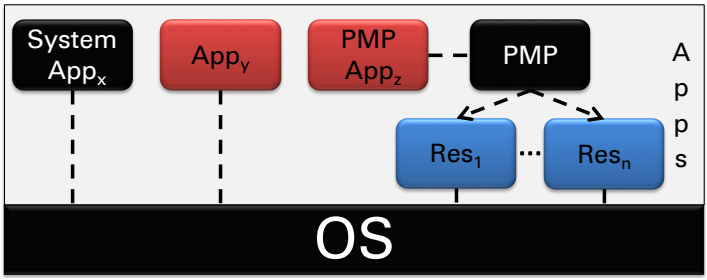


Abbildung 4.14: Implementierung der *PMP* als App (in Anlehnung an [SM14])

Nutzer liegt ein weiterer Vorteil dieser Implementierungsstrategie darin, dass keine Manipulationen an der mobilen Plattform durchgeführt werden. Eine unsachgemäße Veränderung an dieser kann dazu führen, dass das Gerät, auf dem sie läuft, nicht mehr verwendet werden kann oder im schlimmsten Fall sogar Hardwareschäden davontragen kann. Da alle Komponenten der *PMP* allerdings reine Apps sind, kann sie einfach installiert, ausprobiert und ggf. wieder entfernt werden, ohne dass dadurch andere Apps oder gar das System beeinflusst werden. Daher ist die Einstiegshürde für den Nutzer sehr niedrig.

Allerdings kann diese Implementierungsstrategie, gerade weil sie nicht direkt auf Systemebene arbeitet, keinen universellen Schutz bieten. Der Vorteil, dass *Legacy-Apps* ihre Daten direkt von dem Anwendungsframework beziehen können, kann auch von Apps, die grundsätzlich mit der *PMP* zusammenarbeiten, ausgenutzt werden, um Datenzugriffe durchzuführen, die der Nutzer ihnen per *Privacy Rule* eigentlich untersagt hatte. Der Nutzer kann allerdings bei Apps, die ihre Daten von der *PMP* beziehen sollen, darauf achten, dass diese keine zusätzlichen Android-*Permissions* benötigen. Fordern solche Apps dennoch bei der Installation *Permissions* an, so deutet dies auf eine versuchte Unterwanderung der *PMP* hin.

Eine initiale Implementierung der *PMP* gemäß dieser Strategie fand im Rahmen des Studienprojekts von Vetter et al. [VJB+12] statt. Eine vollständig überarbeitete und auf Android 6.0 angepasste Version dieser Implementierungsvariante ist unter [Sta15b] veröffentlicht.

4.3.3 Implementierung als Erweiterung der mobilen Plattform

Eine andere Implementierungsstrategie sieht vor, das neue Datenschutzsystem auf einer tieferen Ebene der mobilen Plattform anzusiedeln, um die Sicherheitsprobleme der in Abschnitt 4.3.2 beschriebenen Strategie ausräumen zu können. Eine Möglichkeit hierfür ist, dass das bestehende Berechtigungssystem der mobilen Plattform vollständig durch das neue Datenschutzsystem ersetzt wird. In diesem Fall muss sichergestellt werden, dass Apps, die auf der Plattform laufen sollen, keine Berechtigungen des alten Systems mehr verwenden; dies hätte ansonsten einen Absturz der betroffenen Apps zur Folge. Um dies gewährleisten zu können, gibt es zwei grundsätzlich unterschiedliche Herangehensweisen, die im Folgenden am Beispiel von Android beschrieben werden:

Manipulation der Berechtigungsinformationen. Eine Android-App wird als Android Application Package (APK)-Datei ausgeliefert. Dabei handelt es sich um ein Archivierungsformat mit Signatur. Vor der Installation der App kann diese Datei daher entpackt, ihr Manifest um jedwede `uses-permission`-Elemente erleichtert und anschließend neu verpackt sowie signiert werden. Zu diesem Zweck kann beispielsweise ein App-Konverter eingesetzt werden (siehe Abschnitt 3.2). Von der Verwendung eines App-Konverters ist in dieser Strategie allerdings abzuraten, da es für diesen eine wesentlich bessere Implementierungsstrategie gibt, die in Abschnitt 4.3.5 detailliert beschrieben wird. Da in der hier beschriebenen Strategie ohnehin die mobile Plattform manipuliert wird, bietet es sich daher an, den Installationsprozess dahingehend abzuändern, dass dieser die `uses-permission`-Einträge ignoriert und die App somit keine *Permissions* erhält. Entzieht man einer App allerdings alle *Permissions*, so führt dies in reinen *Legacy-Apps* dazu, dass diese aufgrund fehlender Berechtigungen häufig abstürzen – *Legacy-Apps* greifen auf Nutzer- und Systemdaten über das Android-Anwendungsframework zu, wofür *Permissions* nötig sind. *System-Apps* sind in dieser Herangehensweise hingegen nicht betroffen, da diese keinen Installationsprozess durchlaufen und diesen somit auch keine Berechtigungen entzogen werden.

Manipulation der Berechtigungsprüfung. Anstelle die Berechtigungsinformationen einer App zu manipulieren, kann auch die Berechtigungsprüfung verändert werden. Wie in Abschnitt 3.1 beschrieben, besteht in Android die

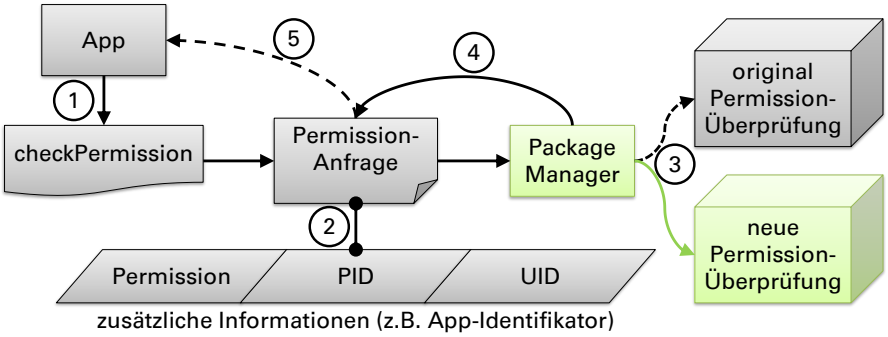


Abbildung 4.15: Manipulation des Berechtigungsprüfungsprozesses (in Anlehnung an [Sta15a])

Berechtigungsprüfung aus einem mehrstufigen Prozess, der durch einen Aufruf der `enforce`-Methode angestoßen wird. In Abbildung 4.15 wird vereinfacht dargestellt, welche Modifikationen an diesem Prozess nötig sind, damit die mobile Plattform für die Berechtigungsprüfung auf ein alternatives Schutzsystem zurückgreift. Zunächst prüft die `checkPermission`-Methode, ob für die Anfrage der App *Permissions* benötigt werden (1). Ist dies der Fall, sendet die Methode eine *Permission*-Anfrage an den `PackageManager`. Die mobile Plattform fügt der Anfrage weitere Informationen über die anfragende App hinzu, damit sich diese eindeutig identifiziert lässt. U. a. zählt hierzu der Process Identifier (PID) und der UID sowie Informationen zu den für die Anfrage benötigten *Permissions* (2). Der Android-Paketmanager prüft anschließend die Berechtigungsanfrage. Für die Überprüfung verwendet dieser weitere Komponenten (3). Da der Paketmanager einerseits die erste Komponente darstellt, an der alle benötigten Informationen vorliegen, und andererseits den Einstiegspunkt für alle Anfragen darstellt, bietet es sich an, diese zentrale Komponente zur Integration eines neuen Schutzsystems zu manipulieren. Eine Möglichkeit hierfür ist es, dass der Paketmanager anstelle der Android-Komponenten zur Berechtigungsüberprüfung einen neuen oder erweiterten Überprüfungsmechanismus zur Entscheidungsfindung aufruft. Liegt dem Paketmanager die Entscheidung über die Berechtigungsanfrage vor, so leitet er entweder die `PERMISSION_GRANTED`- oder die `PERMISSION_DENIED`-Flag an die *Permission*-

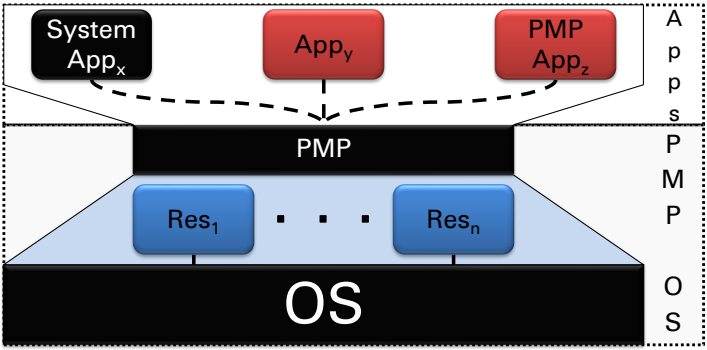


Abbildung 4.16: Implementierung der *PMP* als Erweiterung der App-Plattform (in Anlehnung an [SM14])

Anfrage zurück (4). Wenn die Zugriffsberechtigung nicht erteilt wird, so wird in der anfragenden App eine Sicherheits-Exception geworfen (5).

Die einfachste, dadurch aber sehr performante Manipulation des Paketmanagers besteht darin, dass dieser jede Berechtigungsanfrage direkt ablehnt, wodurch der original Android-Berechtigungsmechanismus obsolet wird. Dies hat allerdings unmittelbar zur Folge, dass alle Apps, inklusive der *Legacy-Apps* und *System-Apps*, das neue Schutzsystem für sämtliche Datenzugriffe, für die eine Berechtigung erforderlich ist, unterstützen müssen.

Damit *System-Apps* in dieser Implementierungsvariante auch unter einem neuen Schutzsystem lauffähig sind, kann der Paketmanager dahingehend erweitert werden, dass dieser zwischen *System-Apps* und Apps von Drittanbietern differenziert. Hierfür reicht es aus, dass dieser den Speicherort der Apps berücksichtigt. *System-Apps* liegen in dem Verzeichnis `/system/app`, während Apps von Drittanbietern stets unter `/data/app` installiert werden. Der Paketmanager hat Zugriff auf die Paketinformationen einer App, zu denen u. a. der Dateipfad der APK-Datei gehört. Wird eine App vom Paketmanager aufgrund des Installationsorts als *System-App* identifiziert, so kann er für jede Berechtigungsanfrage dieser App die `PERMISSION_GRANTED`-Flag zurückgeben.

Nachdem durch eine dieser beiden Manipulationsvarianten sichergestellt ist, dass sämtliche Apps von Drittanbietern über das alte Berechtigungssystem der mobilen Plattform keinen Zugriff auf Nutzer- oder Systemdaten mehr

erlangen, kann in einem zweiten Schritt das neue Schutzsystem in die mobile Plattform integriert werden. In Abbildung 4.16 ist dargestellt, wie die *PMP* in dieser Implementierungsstrategie in die Android-Systemarchitektur eingebettet werden kann. Dabei wird die *PMP* als neue logische Schutzschicht oberhalb des Anwendungsframeworks implementiert.

Konzeptionell müssen alle Apps (also auch *Legacy-Apps* und *System-Apps*) infolgedessen über die *PMP* auf Nutzer- und Systemdaten zugreifen, da die APIs des Anwendungsframeworks für sie nicht mehr sichtbar sind. Praktisch ist dies dadurch sichergestellt, dass die Android-*Permissions* diesen Apps entzogen wurden. Beide obigen Manipulationsvarianten sind allerdings, wie beschrieben, in der Lage, für *System-Apps* Ausnahmen zu definieren, damit diese in ihrer Funktionsweise durch die Einführung des neuen Schutzsystems nicht beeinträchtigt werden.

Durch die Integration der *PMP* auf einer tieferen Architekturebene der Android-Plattform besitzt diese und damit alle *Resources* einen vollen Zugriff auf sämtliche Nutzer- und Systemdaten. Beispielsweise kann die *PMP* direkt in das `/system/app`-Verzeichnis kopiert werden, um den Paketmanager dazu zu veranlassen, ihre Anfragen unabhängig von den tatsächlich vorhandenen *Permissions* zu genehmigen. Dadurch besitzt die *PMP* in dieser Implementierungsvariante implizit sogar System-*Permissions*, ohne dass diese explizit an sie vergeben werden müssen[17].

Der Vorteil an dieser Implementierungsvariante liegt aus Nutzersicht darin, dass *System-Apps* auch weiterhin ohne Anpassungen problemlos funktionieren, während sämtliche Datenzugriffe aller anderen Apps von dem neuen Datenschutzsystem genehmigt werden müssen. Für diese Apps gibt es keinen Weg an dem Datenschutzsystem vorbei, da ihnen alle *Permissions* entzogen bzw. diese ungültig gemacht wurden.

Hierin begründet sich allerdings auch der gravierendste Nachteil für den Nutzer. Ohne diese *Permissions* sind die meisten *Legacy-Apps* nicht mehr lauffähig– zumindest jene, die diese *Permissions* für die Ausführung benötigen, d. h. die auf Daten oder Funktionen zugreifen, für die eine Berechtigung benötigt wird. Auch kann durch die nötige Identifikation der *System-Apps* je nachdem, wie die-

[17] Eine explizite Vergabe von System-*Permissions* an die *PMP* würde auch in dieser Implementierungsvariante an der fehlenden Signatur von Google scheitern.

se realisiert wird, die Performanz der mobilen Plattform beeinträchtigt werden. Für sich genommen ist der Overhead, der durch diese Überprüfung entsteht, minimal, jedoch wird die `enforce`-Methode (und damit auch die Überprüfung) von Android im Hintergrund mit einer sehr hohen Frequenz aufgerufen. In Summe macht sich dadurch selbst ein so geringer, aber häufig anfallender Overhead im Laufverhalten und in der Stabilität des kompletten Systems negativ bemerkbar. Darüber hinaus erfordert diese Implementierungsvariante mehrere tiefgreifende Manipulationen an der mobilen Plattform. Daher ist es erforderlich, dass ein Nutzer die Systempartition seines Smart Devices überschreibt, wenn er das neue Schutzsystem verwenden möchte, was besonders für unerfahrene Nutzer eine hohe Einstiegshürde darstellt.

Die *PMP* wurde im Rahmen der Diplomarbeit von Scholz [Sch13b] nach dieser Implementierungsstrategie prototypisch für ein *Nexus One* auf Basis von Android 2.3.7 realisiert.

4.3.4 Implementierung als alternatives Schutzsystem

Anstelle das bestehende Berechtigungssystem einer mobilen Plattform vollständig zu ersetzen (siehe Abschnitt 4.3.3), kann man ein neues Schutzsystem auch parallel zu diesem einführen. Apps können auf diese Weise sowohl durch das ursprüngliche Berechtigungssystem der mobilen Plattform als auch durch das neue Schutzsystem für den Zugriff auf Nutzer- oder Systemdaten legitimiert werden. Im Gegensatz zu der in Abschnitt 4.3.2 beschriebenen Implementierungsstrategie, in der dies de facto ebenfalls der Fall ist, obliegt bei der Implementierung als alternatives Schutzsystem die Entscheidung darüber, welches System verwendet werden soll, nicht der jeweiligen App, sondern dem neuen Schutzsystem. Insbesondere muss verhindert werden, dass eine App beide Systeme verwenden kann und diese ggf. gegeneinander ausspielen kann.

Aus diesem Grund muss für jede App eindeutig festgelegt sein, ob die Berechtigungsvergabe über das bestehende oder das neue System erfolgen soll. Diese Unterscheidung kann mithilfe einer Überprüfung realisiert werden, ob in einer App Berechtigungen des alten Systems angefordert werden. Dies muss in allen mobilen Plattformen stets an einer zentralen Stelle erfolgen, damit diese schnell und automatisch ausgelesen werden können und diese dem Nutzer beispielsweise im *App-Store* oder bei der Installation der App angezeigt werden

können. Unter Android werden die angeforderten *Permissions* im Manifest der App angegeben. Diese Binary-XML-Datei kann mit einem speziellen Parser nach `uses-permission`-Elementen durchsucht werden. Hierfür stellt Android die `parsePackage`-Methode zur Verfügung, die zur Laufzeit das Manifest auswertet und die relevanten Informationen in einem `Package`-Objekt zusammenfasst. Wird im Manifest eine *Permission* angefordert, so wird das neue Schutzsystem für diese App deaktiviert. Ausschließlich Apps ohne *Permissions* sind für das neue Schutzsystem freigeschaltet.

Bei einer solchen Unterscheidung können zweierlei Fehler auftreten: Ein *False-Positive*-Fehler bezeichnet, dass eine App für das neue Schutzsystem freigeschaltet wird, die dieses aber gar nicht nutzen möchte. Dieser Fehler kann aber nur auftreten, wenn eine App generell keine *Permission* anfordert. Somit wirkt sich dieser Fehler nicht aus, da eine solche App keinen Zugriff auf Daten durchführt, für die Berechtigungen erforderlich sind. Ein *False-Negative*-Fehler bezeichnet, dass das neue Schutzsystem für eine App deaktiviert wird, die dieses jedoch benötigt, um ausführbar zu sein. In diesem Fall hätte die App jedoch zusätzlich *Permissions* angefordert und könnte an dem neuen Schutzsystem vorbei auf Daten zugreifen. Auch wenn eine solche *Permission*-Anforderung ein Versehen sein kann, so kann sie ebenfalls auf einen Täuschungsversuch hindeuten (siehe Abschnitt 4.3.2). Die *False-Negative*-Fehler haben zwar einen sichtbaren Effekt, da die betroffenen Apps ggf. nicht mehr lauffähig sind. Sie sind jedoch aus Sicherheitsgründen tolerabel.

Zusätzlich kann erneut über den Speicherort der App zwischen *System-Apps* und Apps von Drittanbietern unterschieden werden (siehe Abschnitt 4.3.3), um *System-Apps* privilegiert zu behandeln.

Im Fall der *PMP* kann ein solches Identifikationsverfahren noch einfacher realisiert werden und dadurch zeitgleich die Anzahl der *False-Positives* und *False-Negatives* reduzieren. Im Rahmen des Installationsprozesses muss sich eine App bei der *PMP* registrieren, um diese nutzen zu können. Hierfür muss eine Registrierungsaktivität im Manifest der App definiert sein. Wird beim Parsen des Manifests ein entsprechender Eintrag gefunden, so kann die Weitergabe aller `uses-permission`-Einträge, sollten sich solche fälschlicherweise ebenfalls in dem Manifest befinden, an das Android-Berechtigungssystem verhindert werden. Dies kann unmittelbar realisiert werden, da die `parsePackage`-

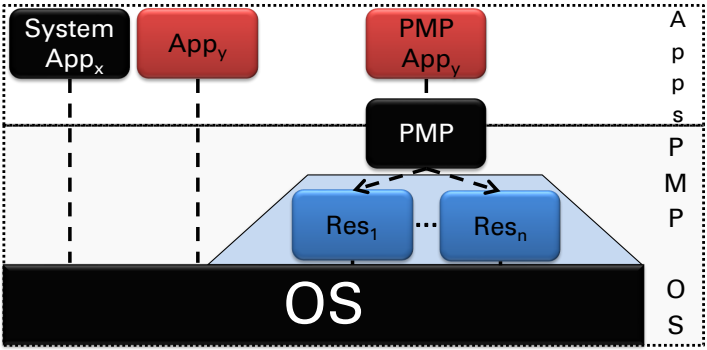


Abbildung 4.17: Implementierung der *PMP* als alternatives Schutzsystem

Methode, mit der ebenfalls eine Liste aller von der App angeforderten *Permissions* (`requestedPermissions`) erstellt werden kann, auch vom Android-Berechtigungssystem genutzt wird. Es genügt daher diese Methode dahingehend zu manipulieren, dass sie für *System-Apps* und *Legacy-Apps* die korrekte, d. h. vollständige *Permission*-Liste und für alle anderen Apps eine leere Liste zurückgibt.

Dadurch sind nur noch solche Apps durch *False-Negative*-Fehler betroffen, die bewusst *Permissions* zusätzlich zur *PMP* verwenden, um diese zu unterwandern. Apps ohne Registrierungsaktivität können die *PMP* nicht nutzen und dürfen daher das originäre Berechtigungssystem nutzen. Auf diese Weise lassen sich für die *PMP False-Positives* gänzlich ausschließen.

In Abbildung 4.17 ist skizziert, welche Auswirkungen diese Implementierungsstrategie auf die Android-Systemarchitektur hat. Auch bei dieser Strategie zieht die *PMP* eine zusätzliche logische Schutzschicht zwischen der Anwendungsschicht und dem Anwendungsframework ein. Alle Apps können infolgedessen nur noch indirekt mit dem Android-System interagieren und müssen konzeptionell alle Anfragen an die *PMP* richten. Speziell für *Legacy-Apps* und *System-Apps* gibt es allerdings Tunnel, durch die ihre Anfragen direkt an die APIs des Anwendungsframeworks ungeprüft durchgeleitet werden. Die Unterscheidung zwischen Apps, die die *PMP* nutzen, und Apps, die die Tunnel nutzen, wird an der Grenze der Schutzschicht durchgeführt. Diese beiden App-Gruppen sind disjunkt.

Auf diese Weise können App-Entwickler frei entscheiden, welches Schutzsystem sie verwenden wollen. Daher ist sichergestellt, dass bei dieser Implementierungsstrategie nicht nur *System-Apps*, sondern auch *Legacy-Apps* weiterhin uneingeschränkt lauffähig sind. Aus Nutzersicht stellt dies einen wertvollen Vorteil dar. Der zusätzliche Overhead, der durch die nötige Identifikation des Berechtigungssystems entsteht, das von einer App verwendet wird, ist vor allem im Fall der *PMP* sehr gering und findet nur einmalig bei der Installation statt.

Allerdings gibt es auch bei dieser Implementierungsstrategie Nachteile. So ist die Identifikation des verwendeten Berechtigungssystems nicht zwangsläufig eindeutig, auch wenn bei der *PMP False-Positives* gänzlich ausgeschlossen werden können (und diese auch keinen wahrnehmbaren Effekt hätten) und die Anzahl der *False-Negatives* minimiert werden kann. Dennoch führt ein *False-Negative* ggf. dazu, dass eine App nicht mehr lauffähig ist. Die betroffenen Apps versuchen dabei die *PMP* zu unterwandern, indem sie zusätzlich ihre Daten direkt von dem Anwendungsframework beziehen. Will ein Nutzer eine solche App dennoch verwenden, z. B. da er dem Entwickler vertraut, so ist dies bei der hier beschriebenen Implementierungsstrategie nicht möglich. Zusätzlich kann die Akzeptanz seitens der App-Entwickler ein Problem bei dieser Implementierungsstrategie darstellen. Da diese zwischen den beiden Schutzsystemen wählen können, ist das wesentlich weniger restriktive Android-Berechtigungssystem aus Entwicklersicht zunächst erheblich verlockender, zumal sie dieses Systems bereits kennen und damit umgehen können. Da die Möglichkeiten, die die *PMP* den Nutzern zum Schutz ihrer Daten bietet, jedoch erheblich umfassender sind, könnte durch die Nutzer, d. h. die Kunden der App-Entwickler, ein Druck dahingehend ausgehen, dass mehr Apps die *PMP* unterstützen.

Der Hauptnachteil dieser Implementierungsstrategie ist allerdings, dass sie für Apps, die das Android-Berechtigungssystem verwenden, keinerlei Verbesserungen darstellt. Daher wurde im Rahmen der Bachelorarbeit von Salsa [Sal14] die *PMP* zwar gemäß dieser Implementierungsstrategie umgesetzt, jedoch wurden weitere Erweiterungen an der *PMP* sowie Verbesserungen an dieser Strategie vorgenommen, um diesen Nachteil auszumerzen. Details zu diesen Erweiterungen und Verbesserungen sind in Abschnitt 4.4 gegeben.

4.3.5 Implementierung mit einem App-Konverter

Ein Hauptproblem für alle Implementierungsstrategien eines neuen Schutzsystems liegt im Umgang mit *Legacy-Apps*, wie die in Abschnitt 4.3.2 bis Abschnitt 4.3.4 beschriebenen Strategien zeigen. Diese Apps versuchen auch weiterhin mit den Berechtigungen des alten Schutzsystems Zugriff auf Nutzer- und Systemdaten zu erhalten. Besonders wenn ein neues Schutzsystem ein erweitertes Modell für Berechtigungsregeln einführt, wie es die *PMP* mit der *PPM* tut, kann das System die alten Berechtigungen entweder nicht verarbeiten oder die verbesserten Schutzmechanismen nicht auf *Legacy-Apps* anwenden. Dies bedeutet, dass jene *Legacy-Apps* entweder weiterhin das alte, unzureichende Berechtigungssystem verwenden oder nicht mehr lauffähig sind. Aus Nutzersicht sind beide Alternativen inakzeptabel. Selbst wenn einem bestehenden System lediglich neue Berechtigungen hinzugefügt werden sollen, führt dies zu Problemen in bestehenden Apps (Abschnitt 4.3.1).

Eine andere Implementierungsstrategie, die gerade den Umgang mit *Legacy-Apps* in den Fokus stellt, führt ein neues Schutzsystem in Kombination mit einem App-Konverter ein. Wie in Abschnitt 3.2 angerissen, dient der App-Konverter dazu, eine App nach datenschutzkritischen Systemaufrufen zu durchsuchen, die spezielle Berechtigungen benötigen. Diese Aufrufe verursachen die obigen Probleme bei der Einführung eines neuen Schutzsystems. Daher ersetzt der App-Konverter im Code diese Aufrufe durch äquivalente Anfragen an das neue Schutzsystem. Dabei entfernt der App-Konverter auch Code, der im neuen Schutzsystem nicht länger benötigt wird und der ggf. Sicherheitsprobleme verursachen könnte, und fügt zusätzlichen Code hinzu, sofern dies für das neue Schutzsystem nötig ist. Beispiele hierfür wären die Eliminierung von `uses-permission`-Elementen aus dem Manifest einer App oder die Ergänzung um Registrierungsaktivitäten für die *PMP*.

Für die Durchführung dieser Schritte muss der App-Konverter direkt auf dem kompilierten Bytecode einer App arbeiten, da deren Quellcode in der Regel nicht zur Verfügung steht. Technisch wird ein App-Konverter aus diesem Grund häufig dadurch realisiert, dass Monitoring-Komponenten in die Apps eingefügt werden. Diese Komponenten fangen alle Berechtigungsanfragen einer App ab und rufen stattdessen entsprechende Methoden des neuen Schutzsystems auf. Anstelle dass das Schutzsystem aktiv Apps überwacht, überwachen diese

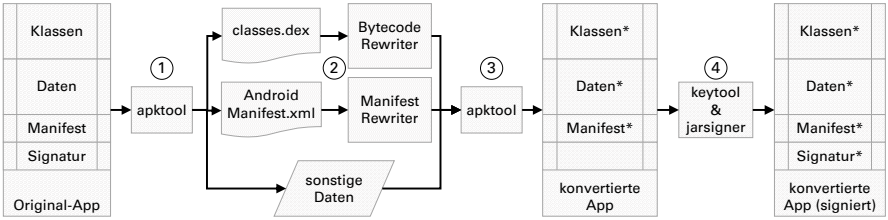


Abbildung 4.18: Werkzeugkette für die Umsetzung eines App-Konverters (in Anlehnung an [Sta15a])

sich mittels dieser Komponenten selbst und arbeiten dem Schutzsystem zu. Nachdem die Monitoring-Komponente eingefügt wurde, muss die veränderte App vom App-Konverter erneut installiert werden. Hao, Singh und Du [HSD13] beschreiben, wie man das von Erlingsson [Erl04] eingeführte Konzept des *Inline-Reference-Monitorings* für die Monitoring-Komponenten nutzen kann.

Für die Realisierung eines derartigen App-Konverters existieren bereits viele Werkzeuge, mit denen der Bytecode einer App verarbeitet werden kann. Abbildung 4.18 stellt eine Werkzeugkette zur Umsetzung eines App-Konverters für Android-basierte Apps dar. Die Original-App liegt dabei im APK-Format, ein auf Java Archive (JAR) basierendes Archivierungsformat, vor. Daher müssen die Inhalte des Archivs zunächst entpackt werden. Das Android-Werkzeug *apktool*[18] kann hierfür verwendet werden. Ein Vorteil dieses Werkzeugs ist, dass es die Bestandteile einer App nicht nur entpackt, sondern ebenfalls deren Binärcode dekompiliert (1).

Anschließend kann das Manifest der App mit jedem beliebigen XML-Parser verarbeitet werden, da auch der Binary-XML-Code des Manifests in regulären XML-Code von *apktool* überführt wird. Somit können beispielsweise `uses-permission`-Elemente daraus entfernt und Registrierungsaktivitäten darin spezifiziert werden. Zur Manipulation von Java-Bytecode gibt es mehrere Werkzeuge, wie beispielsweise *ASM*[19]. Obwohl Android-Apps ebenfalls in Java geschrieben werden, werden sie in Dalvik-Bytecode und nicht Java-Bytecode überführt. Dalvik-Bytecode ist für die Ausführung auf mobilen Plattformen optimiert und unterscheidet sich daher in einigen entscheidenden Punkten

[18] siehe https://ibotpeaches.github.io/Apktool/
[19] siehe http://asm.ow2.org/

von Java-Bytecode. Beispielsweise werden alle Klassen einer App in einer einzigen Dalvik Executable (DEX)-Datei zusammengefasst. Daher können die Java-Bytecode-Manipulationswerkzeuge nicht für die Manipulation von Dalvik-Bytecode verwendet werden. Entweder muss daher der Dalvik-Bytecode zunächst in Java-Bytecode überführt werden (z. B. mit dem Werkzeug *dx*, das mit dem Android-Software Development Kit (SDK) mitgeliefert wird) oder es kommt ein meist experimentelles Dalvik-Bytecode-Manipulationswerkzeug (z. B. *ASMDEX*[20]) zum Einsatz. Mit diesen Manipulationswerkzeugen lassen sich schließlich die Monitoring-Komponenten in die App einbringen (2).

Nachdem die Manipulation durchgeführt wurde, müssen die Bestandteile der App erneut in eine APK-Datei gepackt werden. Bei diesem Schritt kann erneut das *apktool*-Werkzeug verwendet werden (3). Abschließend muss die App erneut signiert werden – durch das Entpacken geht die ursprüngliche Signatur verloren. Hierfür gibt es die Java-Werkzeuge *keytool*[21] und *jarsigner*[22] (4).

Besonders dieser letzte Arbeitsschritt führt zu zwei schwerwiegenden Problemen: Zur Identifikation von Apps verwendet Android u. a. die Signatur. Da der App-Konverter die Signatur ändert, wird die manipulierte App vom System als eine andere App angesehen. Die Konsequenz, die sich hieraus ergibt, ist, dass diese App nicht mehr automatisch über den *App-Store* aktualisiert werden kann. Darüber hinaus sind die Rechte einer App z. T. an deren Signatur gebunden. Beispielsweise können nur Apps mit einer Signatur von Google System-*Permissions* erhalten, oder Apps mit der gleichen Signatur können im gleichen Prozess ausgeführt werden. Da durch den App-Konverter alle Apps die gleiche Signatur erhalten, kann besonders Letzteres von Apps bewusst ausgenutzt werden.

Abgesehen von diesen Android-inhärenten Implementierungsproblemen besitzt diese Implementierungsstrategie zusätzlich die Probleme, die allen Ansätzen, die auf der Manipulation von Apps basieren, immanent sind (siehe Abschnitt 3.2). Da die Bytecode-Manipulation häufig auf Heuristiken beruht, können dabei Fehler auftreten. Des Weiteren stellt diese Manipulation einen Verstoß gegen geltendes Recht dar, da das geistige Eigentum des App-Entwicklers verletzt wird.

[20] siehe http://asm.ow2.org/asmdex-index.html
[21] siehe http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html
[22] siehe http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/jarsigner.html

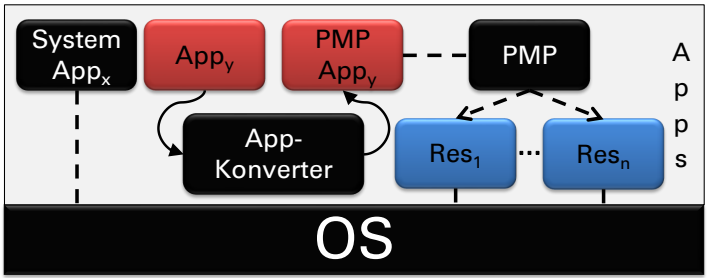


Abbildung 4.19: Implementierung der *PMP* mit einem App-Konverter (in Anlehnung an [SM14])

Trotz dieser rechtlichen Restriktionen, kann eine Werkzeugkette, vergleichbar mit dem von Bartel et al. [BKM+13] vorgeschlagenen Verfahren, für die App-Entwickler angeboten werden, mit der diese ihre eigenen *Legacy-Apps* automatisch an ein neues Schutzsystem anpassen lassen können. In Abbildung 4.19 ist dargestellt, wie die *PMP* nach dieser Implementierungsstrategie in die Android-Systemarchitektur eingebunden werden kann. Die *PMP* kann, genau wie bei der in Abschnitt 4.3.2 vorgestellten Strategie, direkt in der Anwendungsschicht implementiert werden. Die *PMP* ist dadurch nicht in der Lage Apps zu überwachen, da sie keinen Zugriff auf die Sandbox anderer Apps hat. Da alle konvertierten Apps durch die eingebauten Monitoring-Komponenten jedoch mit der *PMP* zusammenarbeiten und aktiv Berechtigungsanfragen an diese schicken, ist dies auch nicht nötig. Die Implementierungsstrategie sieht vor, dass alle Apps von Drittanbietern von einem App-Konverter in Apps, die auf die *PMP* angepasst sind, überführt werden. *System-Apps* werden in dieser Strategie nicht verändert – diese verwenden weiterhin das bestehende, unveränderte Android-Berechtigungssystem, um Komplikationen aufgrund einer veränderten Signatur ausschließen zu können[23].

Für diese Implementierungsstrategie spricht, dass sie die gleiche Sicherheit wie die in Abschnitt 4.3.3 und Abschnitt 4.3.4 vorgestellten Strategien bietet, allerdings ohne dass Manipulationen an der mobilen Plattform nötig sind. Durch die vom App-Konverter in die Apps installierten Monitoring-Komponenten wird sichergestellt, dass Zugriffe auf geschützte Nutzer- oder Systemdaten nur

[23] *System-Apps* besitzen die Google-Signatur, die ihnen mehr Berechtigungen einräumt.

über das neue Schutzsystem erfolgen. Außerdem sind alle *System-Apps* und *Legacy-Apps* unmittelbar kompatibel zu einem nach dieser Strategie realisierten Schutzsystem. Für *Legacy-Apps* ist darüber hinaus nicht nur sichergestellt, dass diese weiterhin funktionsfähig sind, sondern sie unterstützen, nachdem sie konvertiert wurden, direkt das neue Schutzsystem.

Aus Nutzersicht besteht bei dieser Implementierungsstrategie ein großer Nachteil darin, dass der App-Konverter bei jeder Neuinstallation und bei jeder Aktualisierung einer App diese konvertieren muss. Vergisst der Nutzer diesen Schritt durchzuführen, verwendet die App weiterhin das alte, unzureichende Berechtigungssystem. Da sich bei diesem Schritt die Signatur einer App ändert, hat dies direkten Einfluss auf die Rechte einer App. Insbesondere können dadurch alle konvertierten Apps in der gleichen Sandbox ausgeführt werden und aus diesem Grund Daten und sogar Codefragmente von einander nutzen. Eine Veränderung der Signatur kann daher zu Sicherheitsrisiken führen, die für den Nutzer praktisch nicht überschaubar sind.

Ebenfalls darf nicht unterschätzt werden, dass die Manipulation von Bytecode teilweise auf Heuristiken und Vermutungen beruht. Kommt es dabei zu einer falschen Annahme, kann dies gravierende, unvorhersagbare Folgen für die Systemsicherheit und -stabilität haben. Vor allem nach einem Update der mobilen Plattform muss der Entwickler des Schutzsystems daher dafür Sorge tragen, dass der App-Konverter weiterhin kompatibel zu der Plattform ist – ein solches Update kann aber Änderungen an den Bytecode-Mustern, die vom App-Konverter gesucht werden, nach sich ziehen, z. B. wenn mit dem Update neue Berechtigungen eingeführt werden (siehe Abschnitt 4.3.1). Auch muss der App-Konverter an alle Änderungen am Schutzsystem angepasst werden. Für den Nutzer verkompliziert dies die Verwendung des App-Konverters ungemein, da er penibel darauf achten muss, dass er stets die richtige Version verwendet, die sowohl zu seiner mobilen Plattform als auch zu seinem Schutzsystem passt.

Die Implementierung der *PMP* gemäß dieser Strategie wäre zu der in Abschnitt 4.3.2 beschriebenen Strategie identisch. Es könnte daher auf die unter [Sta15b] veröffentlichte Implementierungsvariante zurückgegriffen werden. Auf die Realisierung eines App-Konverters für die *PMP* wurde aufgrund der problematischen rechtlichen Lage jedoch verzichtet.

4.3.6 Vergleich der Implementierungsstrategien

In Tabelle 4.1 werden die wichtigsten Eigenschaften der in Abschnitt 4.3.1 bis Abschnitt 4.3.5 vorgestellten Implementierungsstrategien einander gegenübergestellt. Die letzte Strategie (4.4) wird in Abschnitt 4.4 gesondert besprochen und evaluiert. Die hierbei verwendeten Vergleichskriterien orientieren sich an den in Abschnitt 4.2 aufgestellten Eigenschaften, die für ein Schutzsystem zweckgemäß sind. Es wird analysiert, ob ein Schutzsystem, das nach der jeweiligen Strategie implementiert wurde, in der Lage ist, dem Nutzer ein **Feedback** zu geben (siehe Abschnitt 4.2.1) und **Regeländerungen** zur Laufzeit zuzulassen (siehe Abschnitt 4.2.2). Des weiteren wird untersucht, ob mit dem Schutzsystem auch **Modellanpassungen** an dem Berechtigungsmodell vorgenommen werden können. Hierunter werden beispielsweise Anonymisierungstechniken (siehe Abschnitt 4.2.3) oder die Einbeziehung von Kontextdaten (siehe Abschnitt 4.2.6) subsumiert. Die **Erweiterbarkeit** des Modells, z. B. ob damit auch neue Datenquellen nachträglich unterstützt werden können (siehe Abschnitt 4.2.5), wird ebenfalls analysiert. Auch die Absturzsicherheit (siehe Abschnitt 4.2.4) wird berücksichtigt. Dabei wird für ***System-Apps*** und ***Legacy-Apps*** getrennt betrachtet, ob diese auch unter dem neuen Schutzsystem lauffähig sind.

Zusätzlich zu diesen Eigenschaften wird verglichen, ob jeweils garantiert werden kann, dass Apps nur über das neue Schutzsystem Zugriff auf geschützte Daten oder Funktionen erhalten (**Sicherheit**). Für den Nutzer ist neben der Sicherheit eines solchen Systems aber auch wichtig, dass es einfach zu installieren und einfach zu handhaben ist. Daher wird ebenfalls berücksichtigt, ob die jeweilige Strategie **manipulationsfrei** ist, d. h. ob keine Veränderungen an der mobilen Plattform nötig sind, um das Schutzsystem einzurichten, sowie ob der Nutzer das Schutzsystem verwenden kann, ohne Apps zunächst darauf vorbereiten zu müssen (**automatische Anwendung**).

Bei der Bewertung kommt eine Maßskala mit drei Abstufungen zum Einsatz. Der Füllgrad des dargestellten Kreises gibt an, wie gut eine Eigenschaft unter der jeweiligen Implementierungsstrategie realisiert werden kann.

Bei der *Erweiterung der bestehenden Berechtigungsregeln*-Strategie (siehe Abschnitt 4.3.1) wird deutlich, dass diese für die Implementierung eines neuen Schutzsystems gänzlich ungeeignet ist, da insbesondere das Modell, das den

Feature	Implementierungsstrategie					
	4.3.1	4.3.2	4.3.3	4.3.4	4.3.5	4.4
Feedback	○	◑	◑	●	●	●
Regeländerungen	○	◑	●	◑	●	●
Modellanpassungen	○	◑	●	◑	●	●
Erweiterbarkeit	◑	●	◑	◑	●	●
System-Apps	●	●	◑	●	●	●
Legacy-Apps	◑	●	○	●	●	●
Sicherheit	◑	○	●	◑	◑	●
manipulationsfrei	○	●	○	◑	●	◑
automatische Anwendung	●	●	●	●	○	●

Tabelle 4.1: Vergleich der Implementierungsstrategien (in Anlehnung an [Sta13b])

verwendeten Berechtigungsregeln zugrunde liegt, nicht geändert werden kann. Dadurch können außerdem die Eigenschaften des bestehenden Systems nicht geändert werden (z. B. Feedback an den Nutzer oder Regeländerungen zur Laufzeit). Diese Strategie erlaubt geringfügige Erweiterungen bezüglich der zu schützenden Quellen, da hierfür zwar neue Berechtigungsarten eingeführt werden können, deren Umsetzung aber kein Teil der gelieferten Erweiterung ist. Da *System-Apps* in der Regel vom Anbieter der mobilen Plattform gewartet werden, sind diese stets an Veränderungen der Plattform angepasst und werden daher unmittelbar unterstützt. *Legacy-Apps* hingegen können durch die neuen Berechtigungsregeln betroffen sein, was zu Abstürzen führen kann. Obwohl die neuen Regeln direkt für alle Apps angewandt werden, bieten sie nur einen unwesentlich verbesserten Schutz. Für die Einbringung der neuen Regeln muss das System vollständig ausgetauscht werden.

Die *Implementierung als App*-Strategie (siehe Abschnitt 4.3.2) gestattet es, ein vollständig neues Modell für Berechtigungsregeln, wie beispielsweise das

PPM (siehe Abschnitt 4.1) einzuführen. Dadurch sind in dieser Implementierungsstrategie auch ein Feedback an den Nutzer oder Regeländerungen zur Laufzeit möglich, allerdings gilt das neue Modell sowie die damit verbundenen Eigenschaften nicht für *System-Apps* oder *Legacy-Apps*. Da alle Komponenten des neuen Schutzsystems in der Anwendungsschicht angesiedelt sind, bedarf es keiner Systemmanipulation und es können problemlos weitere Komponenten, genau wie Apps, jederzeit nachinstalliert werden. Da sowohl *System-Apps* als auch *Legacy-Apps* weiterhin das bestehende Berechtigungssystem nutzen können, wird deren Funktionalität durch das neue Schutzsystem nicht beeinflusst; jedoch bietet es für solche Apps auch keinen Schutz. Jede App kann selbst entscheiden, über welches System sie Zugang zu geschützten Daten erhalten möchte, wodurch das neue Schutzsystem unterwandert werden kann. Für die Apps, die das neue Schutzsystem verwenden wollen, steht es nach der Installation unmittelbar zur Verfügung.

Ein Schutzsystem, das nach der *Implementierung als Erweiterung der mobilen Plattform*-Strategie (siehe Abschnitt 4.3.3) implementiert wurde, kann ebenfalls ein neues Berechtigungsmodell einführen und ermöglicht damit Feedback an den Nutzer oder Regeländerungen zur Laufzeit. Jedoch wird vom dem neuen Schutzsystem nur noch dieses veränderte Berechtigungsmodell unterstützt. Daher sind weder *System-Apps* noch *Legacy-Apps* darunter lauffähig. Allerdings können für *System-Apps* Ausnahmen definiert werden, damit diese auch weiterhin, wenn auch vollständig unreglementiert, funktionieren. Feedback kann ein auf diese Weise implementiertes System nur für Apps geben, die ebenfalls das neue Berechtigungsmodell verwenden. Da das Schutzsystem tiefer in die Systemarchitektur eingebunden ist, ist es unmittelbar in der Lage alle Apps zu kontrollieren, wodurch vollständig ausgeschlossen werden kann, dass das System von einer App unterwandert wird. Hierfür sind allerdings tiefgreifende Manipulation an der mobilen Plattform nötig. Erweiterungen können zwar installiert werden, allerdings nur auf einer höheren (und damit weniger geschützten) Ebene der Systemarchitektur.

In der *Implementierung als alternatives Schutzsystem*-Strategie (siehe Abschnitt 4.3.4) kann ebenfalls ein neues Berechtigungsmodell eingeführt werden. Da mit dem neuen Schutzsystem eine logische Schutzschicht zwischen der Anwendungsschicht und der mobilen Plattform eingezogen wird, laufen

die Berechtigungsanfragen aller Apps über das System. Daher kann ein so implementiertes Schutzsystem den Nutzer über alle diese Anfragen informieren. Regeländerungen können zwar zu Laufzeit vorgenommen werden, diese gelten allerdings nicht für *System-Apps* oder *Legacy-Apps*. Genau wie bei der *Implementierung als Erweiterung der mobilen Plattform*-Strategie können Erweiterungen nachinstalliert werden; für diese gelten jedoch nicht die gleichen Sicherheitseigenschaften, da diese in einer höheren Ebene der Systemarchitektur angesiedelt werden. Da für *System-Apps* und *Legacy-Apps* weiterhin das alte Berechtigungssystem zur Verfügung steht, wird deren Funktionalität durch das neue Schutzsystem nicht eingeschränkt. Die verbesserte Sicherheit, die durch das neue Schutzsystem eingeführt wird, gilt allerdings für diese Apps auch nicht. Zur Einrichtung der logischen Schutzschicht sind zwar Anpassungen an der mobilen Plattform nötig, diese sind allerdings geringfügig und das bestehende Berechtigungssystem bleibt erhalten. Das Schutzsystem führt bei dieser Implementierungsvariante eine Komponente ein, mit der für jede App entschieden werden kann, ob das neue oder das alte Schutzsystem verwendet werden soll. Dieses wird automatisch für jede App angewandt und es sind keine App-Manipulationen nötig.

Die Implementierungsstrategie *Implementierung mit einem App-Konverter* (siehe Abschnitt 4.3.5) führt ein neues Berechtigungsmodell ein, das mittels eines App-Konverters auf alle Apps angewandt wird. Daher kann ein Schutzsystem, das dieser Strategie folgt, auch Feedback zu allen Apps geben sowie Regeländerungen zur Laufzeit ermöglichen. Durch den App-Konverter ist sichergestellt, dass alle Apps sich selbst kontrollieren und ihre Anfragen direkt an das neue Schutzsystem richten. Daher kann das Schutzsystem auf der Anwendungsebene angesiedelt werden und ist somit vollständig erweiterbar. Der App-Konverter ist in der Lage alle Apps zu konvertieren, wodurch *System-Apps* und *Legacy-Apps* nicht nur weiterhin lauffähig sind, sondern sogar das neue Schutzsystem verwenden. Allerdings wird die Sicherheit bei dieser Strategie dadurch eingeschränkt, dass der App-Konverter Entscheidungen treffen muss, die auf Heuristiken beruhen. Hierbei können Fehler auftreten, die die Sicherheit des Systems beeinträchtigen, indem entweder Datenzugriffe in einer App übersehen und somit nicht reglementiert werden oder durch die Konvertierung zusätzliche Sicherheitslücken in eine App eingebracht werden. Manipulationen

an der mobilen Plattform sind bei dieser Implementierungsstrategie nicht nötig, jedoch verwenden Apps das neue Schutzsystem erst, nachdem diese konvertiert wurden. Vergisst der Nutzer diesen Schritt, so wird die betreffende App nicht kontrolliert und der Schutz der Nutzer- und Systemdaten ist gefährdet. Des Weiteren verstößt der Einsatz eines App-Konverters gegen geltendes Recht, womit diese Strategie in der Praxis für die Implementierung eines Schutzsystems nicht in Frage kommt.

Bei der Evaluation wird offensichtlich, dass es für den Nutzer keine optimale Implementierungsstrategie gibt, sondern nur lokale Optima für bestimmte Einsatzzwecke. Insgesamt gehen die Strategien 4.3.4 und 4.3.5 als Sieger aus diesem Vergleich hervor, da beide ein neues Berechtigungsmodell einführen können, unter beiden sowohl *System-Apps* als auch *Legacy-Apps* weiterhin lauffähig sind und beide mit wenigen bzw. keinen Manipulationen an der mobilen Plattform installiert werden können. Da die rechtliche Problematik des App-Konverters die Strategie 4.3.5 ausschließt, wird im Folgenden mit dem *PMP-Gatekeeper* eine Erweiterung für die *PMP* vorgestellt, die der *Implementierung als alternatives Schutzsystem*-Strategie folgt, aber den Nachteil dieser Strategie, dass für *System-Apps* und *Legacy-Apps* keine erweiterten Schutzfunktionen existieren, ausmerzt.

4.4 Der PMP-Gatekeeper

Der *PMP-Gatekeeper* ist eine Erweiterungskomponente für die *PMP*, deren Hauptaufgabe darin besteht, für alle auf einem Smart Device installierten Apps eindeutig bestimmen zu können, ob es sich dabei um eine *System-App*, eine *Legacy-App* oder eine App, die die *PMP* für ihre Datenzugriffe nutzt (im Nachfolgenden als PMP-*kompatible App* bezeichnet), handelt. Damit bietet der *PMP-Gatekeeper* zunächst genau die Funktionalität, wie sie in der *Implementierung als alternatives Schutzsystem* (siehe Abschnitt 4.3.4) benötigt wird. Für *Legacy-Apps* erweitert und verbessert der *PMP-Gatekeeper* zusätzlich das von der mobilen Plattform verwendete Berechtigungssystem, so dass diese Apps (und auf Wunsch selbst *System-Apps*) vom Nutzer besser kontrolliert und reglementiert werden können. Obwohl der *PMP-Gatekeeper* sich damit perfekt in die *Implementierung als alternatives Schutzsystem*-Strategie einfügt und diese

optimal ergänzt, kann er grundsätzlich auch für jede der in Abschnitt 4.3.2 bis Abschnitt 4.3.5 vorgestellten Implementierungsstrategien gewinnbringend eingesetzt werden. Im Folgenden ist der Aufbau des *PMP-Gatekeepers* sowie dessen Funktionsweise näher veranschaulicht.

Wie in Abschnitt 4.3.4 beschrieben, erfolgt die Identifikation des App-Typs über den Paketmanager, genauer mithilfe der `getPackageInfo`-Methode. Wenn diese Methode eine Registrierungsaktivität der *PMP* ermittelt, überprüft der *PMP-Gatekeeper* zusätzlich, ob die App auch ein *AIS* definiert. Diese Überprüfung sichert die Identifikation des App-Typs ab[24]. Werden beide Komponenten in einer App gefunden, so klassifiziert der *PMP-Gatekeeper* diese als eine PMP-*kompatible App* und löscht alle *Permission*-Elemente aus deren Manifest. Schlägt eine der beiden Überprüfungen fehl, so ist die App nicht kompatibel zur *PMP*. Es handelt sich demnach um eine *System-App* oder eine *Legacy-App*.

Die Unterscheidung zwischen *System-Apps* und *Legacy-Apps* wird jedoch nicht nur auf Basis des Speicherorts getroffen (siehe Abschnitt 4.3.3), da ein App-Entwickler ansonsten das Schutzsystem aushebeln könnte, wenn es ihm gelingt, seine App in das Systemverzeichnis zu verschieben. Daher überprüft der *PMP-Gatekeeper* zusätzlich, ob eine App die sogenannte `FLAG_SYSTEM`-Flag besitzt. Diesen Statusindikator können nur Apps tragen, die von Google signiert wurden. Durch diese doppelte Überprüfung können auch *System-Apps* und *Legacy-Apps* vom *PMP-Gatekeeper* eindeutig und fehlerfrei unterschieden werden.

Basierend auf dieser Klassifikation (siehe Abbildung 4.20), werden Berechtigungsanfragen entweder an die *PMP* (PMP-*kompatible App*) oder an ein spezielles Berechtigungsmanagementsystem des *PMP-Gatekeepers* für *Legacy-Apps* weitergeleitet. Berechtigungsanfragen von *System-Apps* werden ohne weitere Überprüfung gestattet und an die mobile Plattform weitergeleitet. Da allerdings Plattformanbieter mit ihrer mobilen Plattform häufig auch Apps von Drittanbietern ausliefern (z. B. die *Facebook*-App) und diese ebenfalls als *System-App* deklarieren, können Nutzer im *PMP-Gatekeeper System-Apps* auswählen, die zu *Legacy-Apps* herabgestuft werden sollen, wodurch diese ebenfalls vom *PMP-Gatekeeper* überwacht werden. In Abbildung 4.21 ist die logische Integration des *PMP-Gatekeepers* in die *PMP* sowie das Zusammenspiel

[24] Für diese Überprüfung kann auf den *AssetManager* von Android zurückgegriffen werden.

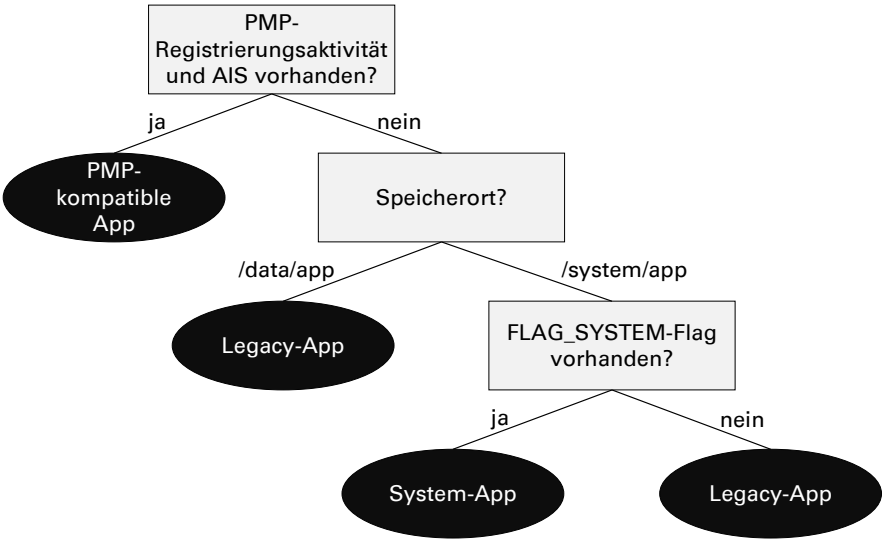


Abbildung 4.20: Entscheidungsbaum zur Klassifikation des App-Typs im *PMP-Gatekeeper*

zwischen dem *PMP-Gatekeeper* und den restlichen involvierten Komponenten dargestellt.

Da die Berechtigungsanfragen von *Legacy-Apps* respektive *System-Apps* auf dem *Permission*-Modell von Android beruhen, kann der *PMP-Gatekeeper* für diese Apps kein weitreichendes Berechtigungsmanagement anbieten, wie es die *PMP* zu leisten im Stande ist. Vielmehr führt der *PMP-Gatekeeper* für *Legacy-Apps* ein Berechtigungssystem ein, das mit den *Runtime Permissions* in Android 6.0 vergleichbar ist, d. h. der Nutzer kann zur Laufzeit *Legacy-Apps* die von diesen angeforderten *Permissions* entziehen. Im Gegensatz zu den *Runtime Permissions* bietet der *PMP-Gatekeeper* allerdings vier entscheidende Vorteile:

(1) Der *PMP-Gatekeeper* kann einer App jede beliebige *Permission* entziehen, während das Konzept der *Runtime Permissions* nur für eine vorgegebene Teilmenge der existierenden *Permissions* gilt.

(2) Der *PMP-Gatekeeper* kann auch die Berechtigungen von *System-Apps* einschränken.

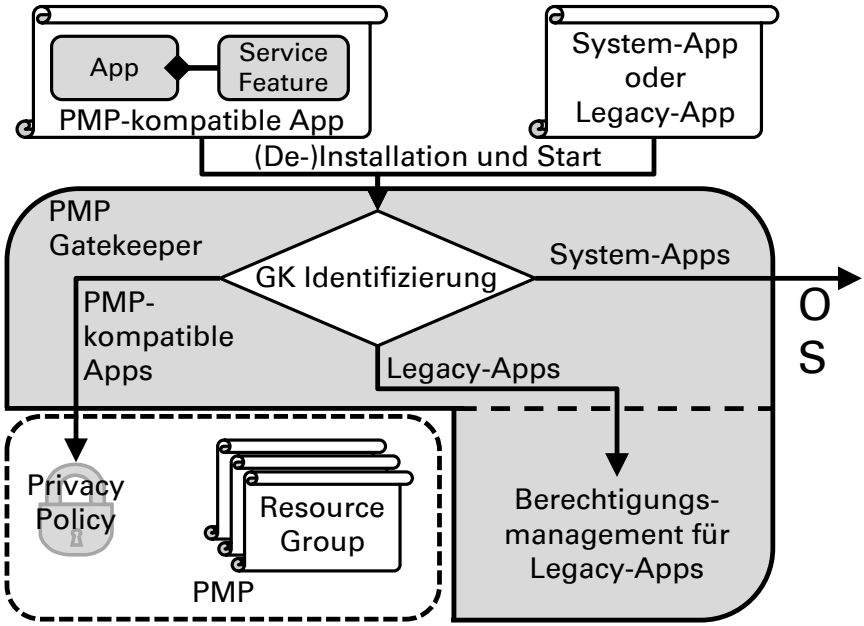


Abbildung 4.21: Logische Integration des *PMP-Gatekeepers* in die *PMP* (in Anlehnung an [Sta15a])

(3) Der *PMP-Gatekeeper* benötigt keine Anpassungen an *Legacy-Apps*; Google führte hingegen für die *Runtime Permissions* einen neuen Anfrageprozess für Berechtigungen ein, der von App-Entwicklern eingehalten werden muss.

(4) Der *PMP-Gatekeeper* ist auf allen Android Versionen (ab Android 2.1) lauffähig, während die *Runtime Permissions* erst ab Android 6.0 nutzbar sind.

Um ein solches Managementsystem technisch realisieren zu können, reicht eine Manipulation der *Permission*-Einträge im Manifest der Apps nicht aus. Das Manifest wird unter Android nur einmalig bei der Installation der App ausgewertet und die Metadaten der App werden in einem geschützten Speicherbereich des Systems abgelegt. Dort findet auch die Berechtigungsprüfung statt. Der Inhalt dieses Speichers wird nur persistiert, wenn das System her-

```
<shared-user name="com.google.android.apps.translate"
             userId="10065">
  ...
  <perms>
    <item name="android.permission.BROADCAST_SMS" />
    <item name="android.permission.CAMERA" />
    <item name="android.permission.INTERNET" />
    <item name="android.permission.READ_PROFILE" />
    <item name="android.permission.READ_SMS" />
    <item name="android.permission.USE_CREDENTIALS" />
    ...
  </perms>
</shared-user>
```

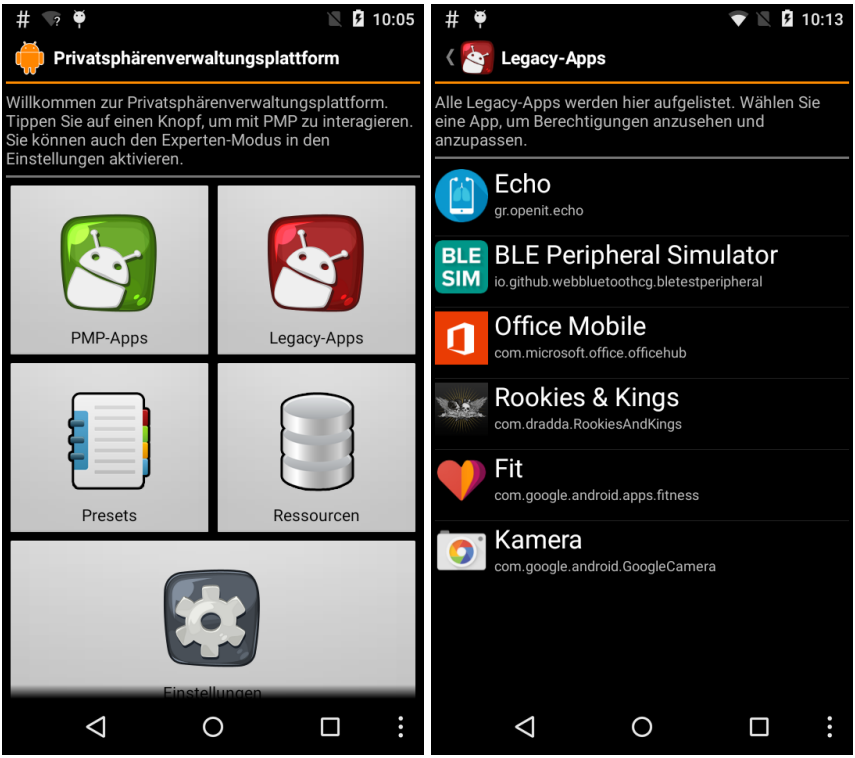
Quelltext 4.6: *Permissions* der *Translate*-App (Auszug aus `packages.xml`)

untergefahren wird. Der Aufbau dieser Sicherungskopie (*packages.xml*) ist in Quelltext 4.6 in Ausschnitten für die Android-App *Translate* dargestellt[25]. Im Rahmen des ebenfalls geschützten Boot-Prozesses werden diese Dateien ausgelesen und die Daten in den geschützten Speicher übertragen. Anschließend sind die persistenten Kopien ungültig.

Um auf den geschützten Speicherbereich zugreifen zu können, sind weitreichende Eingriffe in die mobile Plattform erforderlich. Die Implementierungsstrategie zur *Implementierung als alternatives Schutzsystem* versucht jedoch die Manipulationen an der mobilen Plattform auf ein Mindestmaß zu beschränken. D. h. ohne eine solche Manipulation können auf diesem Weg einer App die *Permissions* nicht entzogen werden.

Daher verfolgt der *PMP-Gatekeeper* eine andere Strategie. Genau wie das original Android-System erzeugt er ein Regelwerk für alle von ihm kontrollierten *System-Apps* sowie alle *Legacy-Apps*, das er in seinem privaten Speicher hält und nur zu einigen wenigen Zeitpunkten (z. B. wenn das System heruntergefahren wird) in eine Datei ausschreibt. Da sowohl die persistente als auch die flüchtige Version des Regelwerks vom *PMP-Gatekeeper* erzeugt werden, hat

[25] Hierbei ist anzumerken, dass das Datenschema dieser Datei nicht einheitlich ist und sich von App zu App vollständig unterscheiden kann. Auch werden die *Permission*-Informationen mancher Apps in andere Dateien ausgelagert, ohne dass es dafür ersichtliche Gründe gibt.

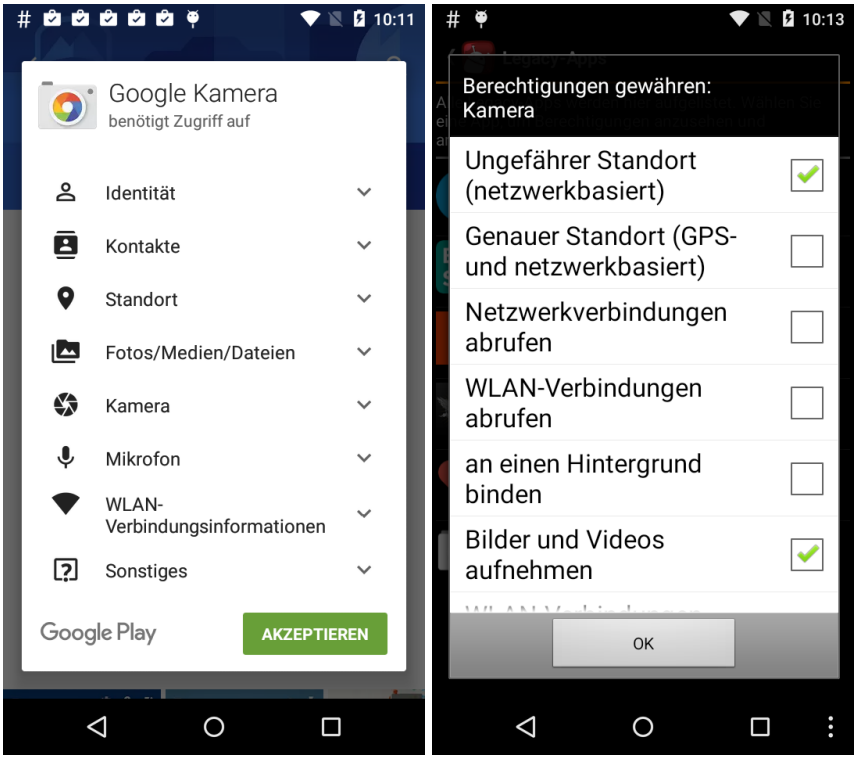


(a) Erweiterter Hauptbildschirm der *PMP* (b) Übersicht der durch den *PMP-Gatekeeper* kontrollierten Apps

Abbildung 4.22: Erweiterte GUI der *PMP*

dieser jederzeit volle Zugriffsrechte auf beide Versionen. Bei der Installation einer App ermittelt der *PMP-Gatekeeper* alle angeforderten *Permissions* und fügt diese dem Regelwerk hinzu. Per Default werden einer vom *PMP-Gatekeeper* überwachten App zunächst alle *Permissions* gewährt. Bei Bedarf kann der Nutzer allerdings zur Laufzeit beliebige *Permissions* einer App deaktivieren bzw. diese später wieder aktivieren.

Hierfür erweitert der *PMP-Gatekeeper* den Hauptbildschirm der *PMP*, wie in Abbildung 4.22a dargestellt. Im neu hinzugekommenen Menü für *Legacy-Apps* werden alle vom *PMP-Gatekeeper* überwachten Apps – also auch die vom Nutzer selektierten System-Apps – aufgeführt (siehe Abbildung 4.22b). Wählt



(a) Berechtigungsanzeige von Android (b) Berechtigungsanzeige des *PMP-Gatekeepers*

Abbildung 4.23: Berechtigungsmanipulation von Drittanbieter-Apps

der Nutzer eine der Apps aus, zeigt ihm der *PMP-Gatekeeper* alle von der App angeforderten *Permissions* an. Android hingegen gibt dem Nutzer bei der Installation einer App nur eine stark aggregierte Sicht auf deren *Permissions*. Einige *Permissions* werden dabei überhaupt nicht aufgeführt, da diese von Google als *unkritisch* betrachtet werden[26] (siehe Abbildung 4.23a). Der *PMP-Gatekeeper* zeigt im Gegensatz dazu alle real vorhandenen *Permissions* einer App an (siehe Abbildung 4.23b). Über diese Ansicht kann der Nutzer so feingranular, wie es das Android-*Permission*-System zulässt, einer App Berechtigungen entziehen.

[26] Hierzu zählt u. a. auch die `INTERNET`-*Permission*, d. h. die Berechtigung, dass eine App unbemerkt jederzeit beliebige Nutzerdaten an Dritte weiterleiten darf!

Diese Einstellungen speichert der *PMP-Gatekeeper* in seinem Regelwerk. Da der *PMP-Gatekeeper* logisch die einzige Schnittstelle zur mobilen Plattform und damit zum Anwendungsframework darstellt, kann er sämtliche Berechtigungsanfragen von Apps abfangen und gemäß den Vorgaben des Nutzers, d. h. gemäß des *PMP-Gatekeeper*-Regelwerks, die Anfrage zulassen oder ablehnen. Da eine Ablehnung eine Sicherheits-Exception in der App auslösen würde, die ggf. darin nicht behandelt wird und somit unweigerlich zum Absturz der App führen würde, versorgt der *PMP-Gatekeeper* in diesem Fall die App mit leeren Datensätzen[27].

Für die Realisierung des *PMP-Gatekeepers* sind daher im Wesentlichen zwei Komponenten erforderlich: Spezielle Schnittstelle zur *PMP* und eine Erweiterung der mobilen Plattform. Diese werden im Folgenden näher beschrieben.

Schnittstelle zur *PMP*. Das Android-Anwendungsframework verwendet ein Ereignissystem zur Kommunikation mit den tieferen Schichten der mobilen Plattform. Über das Ereignissystem wird das Anwendungsframework über eingehende Short Message Service (SMS)-Nachrichten, verfügbare WLANs oder Änderungen am Verbindungsstatus zu Bluetooth-Geräten in Kenntnis gesetzt. Diese Systemnachrichten, die sogenannten *Broadcasts*, können aber auch von jeder App mitgelesen werden. Hierzu muss eine App lediglich einen zu dem jeweiligen Ereignis passenden *BroadcastReceiver* in ihrem Manifest definieren und implementieren. Diese Systemnachrichten informieren allerdings nicht nur über hardwarebasierte Ereignisse, sondern auch über softwareseitige. Beispielsweise meldet der Paketmanager, wann immer eine App installiert oder entfernt wird.

Die *PMP* besitzt bereits einen *InstallReceiver* und einen *UninstallReceiver*, die auf die Installation respektive Deinstallation von Apps warten, und anschließend den erweiterten Installationsprozess der *PMP* starten (siehe Abbildung 4.5b) bzw. die zu der App gespeicherten Informationen, Einstellungen und *Privacy Rules* löschen. Der *PMP-Gatekeeper* kann sich daher direkt in den *InstallReceiver* einklinken, um darin seine Klassifikation zwischen PMP-*kompatible Apps*, *Legacy-Apps* und *System-Apps* durchzuführen. Abbildung 4.24 zeigt den internen Kontrollfluss für den Installations- respektive Deinstallationsprozess.

[27] Je nachdem welche Art von Daten eine App anfragt, handelt es sich dabei um `null`-Werte oder das Element `0`.

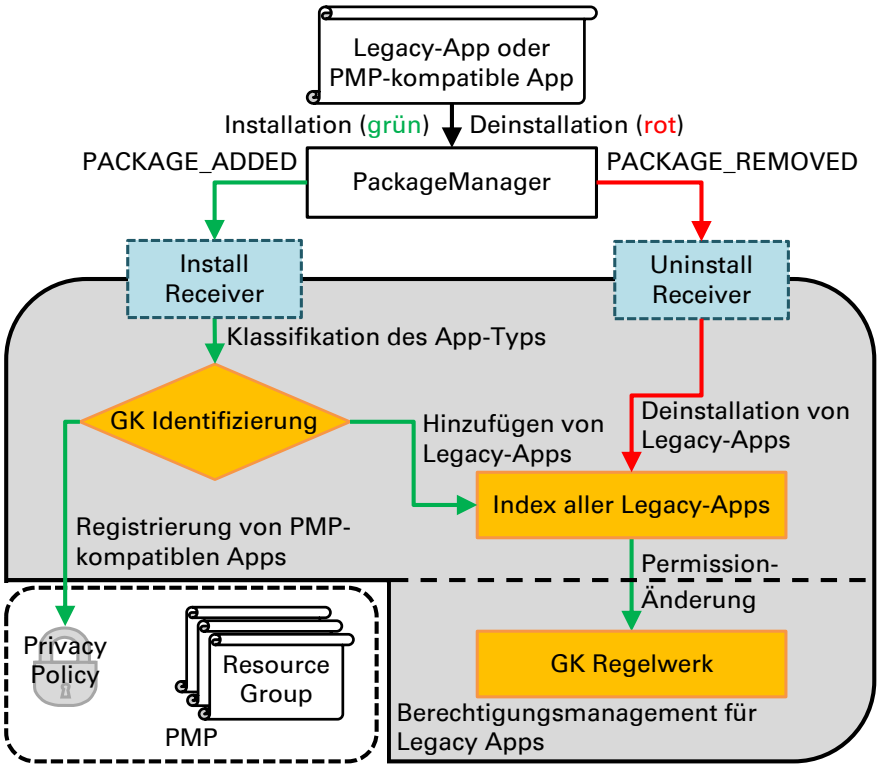


Abbildung 4.24: Kontrollfluss des *PMP-Gatekeepers* (GK) bei der Installation bzw. Deinstallation von Apps (der *PMP-Gatekeeper* ist grau, neue Komponenten orange und erweiterte Komponenten blau dargestellt)

Der Paketmanager sendet einen `PACKAGE_ADDED`-*Broadcast* aus, sobald eine App installiert wird. Der *InstallReceiver* reagiert auf dieses Ereignis, indem er die Informationen zu der App an den *PMP-Gatekeeper* weiterleitet. Dieser identifiziert gemäß den obigen Klassifikationsmerkmalen, um welchen App-Typ es sich handelt. Bei einer PMP-*kompatiblen App* leitet er den Installations- und Registrierungsdialog der *PMP* ein. Für *Legacy-Apps* liest er alle angeforderten *Permissions* aus und erzeugt dafür Einträge in sein Regelwerk. Zusätzlich wird über den App-Name inklusive der Paket-Angaben eine Indexstruktur erzeugt, um schnell auf die zugehörigen Regeln zugreifen zu können.

Der PACKAGE_REMOVED-*Broadcast* stößt analog dazu den Deinstallationsprozess an. In dieser Systemnachricht ist der Name der App, jedoch nicht die App selbst enthalten. Der *PMP-Gatekeeper* kann daher die App nicht mehr analysieren, um herauszufinden, um welchen App-Typ es sich handelt. Für *Legacy-Apps*, die vom *PMP-Gatekeeper* reglementiert werden, existiert allerdings ein Eintrag in der Indexstruktur, selbst wenn die App keine *Permissions* anfordert. Darüber kann er in der Indexstruktur nach der App suchen und ggf. alle gespeicherten Informationen zu dieser App löschen, sollte es sich um eine *Legacy-App* handeln. Dieser Schritt ist nicht zwingend erforderlich, aber er sorgt dafür, dass das Regelwerk des *PMP-Gatekeepers* nicht auswuchert.

Für diese beiden Schritte müssen *System-Apps* nicht berücksichtigt werden, da diese vom Nutzer weder installiert noch entfernt werden können. Allerdings selbst wenn der Nutzer, beispielsweise mittels *Root*-Rechten eine solche App löscht, ohne den Paketmanager zu verwenden, stellt dies kein Problem für den *PMP-Gatekeeper* dar. Der Speicherplatz, den die Regeln einer einzigen App im Regelwerk belegen, ist vernachlässigbar, und da die Anzahl der *System-Apps* ebenfalls marginal ist, hätte selbst die Löschung aller *System-Apps* keine wahrnehmbare Auswirkung auf die Performanz des *PMP-Gatekeepers*.

Erweiterung der mobilen Plattform. Während für diese Schnittstelle zur *PMP* keine Manipulationen an der mobilen Plattform nötig sind, muss zur Durchsetzung des Regelwerks das Anwendungsframework der mobilen Plattform logisch von der Anwendungsschicht getrennt werden. Ansonsten kann kein zuverlässiger Schutz gewährleistet werden (siehe Abschnitt 4.3.6). Wie in Abschnitt 3.2 dargelegt, handelt es sich bei der Berechtigungsanfrage in Android um einen mehrstufigen Prozess. Die Strategie des *PMP-Gatekeepers* ist es, diesen Prozess so früh wie möglich, d. h. sobald alle von ihm benötigten Informationen vorliegen, zu unterbrechen, um keinen unnötigen Overhead zu erzeugen. Alles was der *PMP-Gatekeeper* über eine App wissen muss, ist deren Name sowie deren Paket-Angaben, da darüber sein Regelwerk indiziert ist. Diese Angaben liegen im Android-`ActivityManager` vor, der die `checkPermission`-Methode verarbeitet (siehe Abbildung 4.15). Daher klinkt sich der *PMP-Gatekeeper* in diese Methode ein.

Die dafür benötigten Änderungen sind minimal. Quelltext 4.7 zeigt den modifizierten Quellcode. Zunächst prüft der *PMP-Gatekeeper*, ob die App die

```
1 checkPermission(String permission, int pid, int uid) {
2   if (permission == null)
3     return PackageManager.PERMISSION_DENIED;
4   if (checkComponentPermission(permission,pid,uid,-1) ==
    ↪  PackageManager.PERMISSION_GRANTED) {
5         String[] packageNames =
          ↪  PackageManager.getPackagesForUid(uid);
6         return checkGatekeeper(packageNames, permission);
7   } else
8     return PackageManager.PERMISSION_DENIED;
9 }
```

Quelltext 4.7: Code-Erweiterungen der `checkPermission`-Methode

angeforderten *Permission* überhaupt erhalten kann (Zeile 4). Beispielsweise dürfen keine System-*Permissions* an Apps vergeben werden, die nicht von Google signiert wurden, selbst wenn der Nutzer dies genehmigt. Dies würde ein zu großes Sicherheitsrisiko darstellen. Wenn die App theoretisch die *Permission* erhalten kann, wird der *PMP-Gatekeeper* aktiv und stößt mit der `checkGatekeeper`-Methode seinen eigenen Überprüfungsprozess an (Zeile 6).

Dieser Prozess ist in Abbildung 4.25 dargestellt. Nachdem der *PMP-Gatekeeper* die anfragende App klassifizieren konnte, leitet er die Anfrage entsprechend weiter. Anfragen von PMP-*kompatiblen Apps* werden ausschließlich von der *PMP* verarbeitet. Anfragen von *System-Apps* werden, sofern der Nutzer die anfragende App nicht als *Legacy-App* herabgestuft hat, stets genehmigt und an die mobile Plattform weitergeleitet. Anfragen von *Legacy-App* werden vom *PMP-Gatekeeper* geprüft. Hierzu greift er über die Indexstrukturen auf sein Regelwerk zu und schaut nach, ob die angefragte *Permission* vom Nutzer deaktiviert wurde. Anschließend genehmigt er entweder die Aktion der App oder leitet dieser einen leeren Datensatz zurück.

Wird die *PMP* in Kombination mit dem *PMP-Gatekeeper* gemäß der Strategie *Implementierung als alternatives Schutzsystem* implementiert, so stellt dies die optimale Vorgehensweise dar. Wie in Tabelle 4.1 zu sehen ist, erfüllt sie nahezu alle Bewertungskriterien vollständig. Grundsätzlich besitzt sie alle Eigenschaften der *Implementierung als alternatives Schutzsystem*-Strategie.

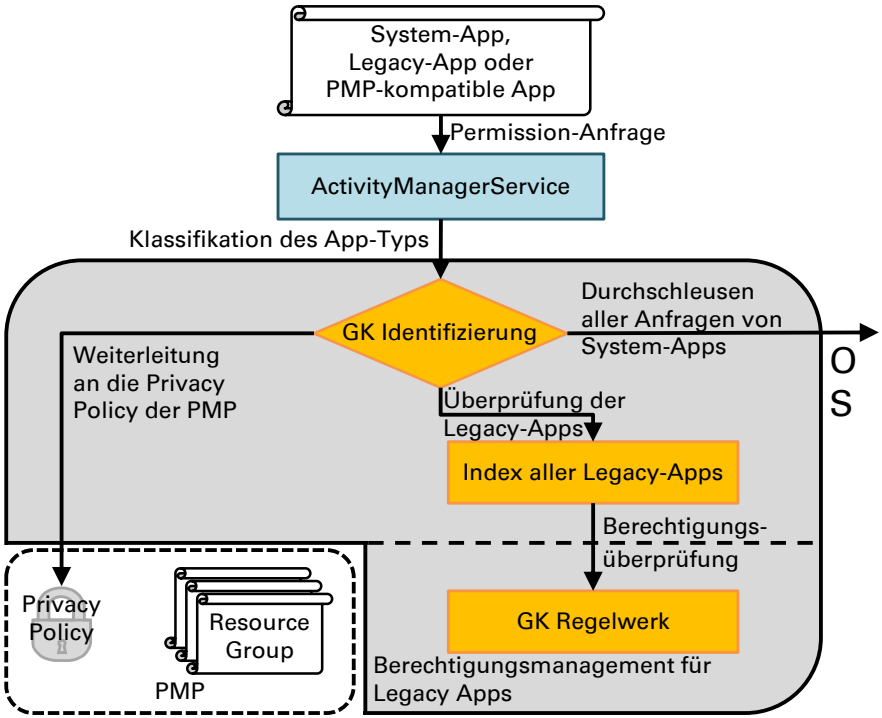


Abbildung 4.25: Kontrollfluss des *PMP-Gatekeepers* bei einer Berechtigungsanfrage (neue Komponenten sind orange und erweiterte Komponenten blau dargestellt)

Allerdings gestattet sie, dank des *PMP-Gatekeepers* auch für *Legacy-Apps* und sogar für *System-Apps*, dass Berechtigungen durch den Nutzer dynamisch zur Laufzeit entzogen oder genehmigt werden können. Die *Permissions* können selbst für *Legacy-Apps* auf feingranularer Ebene vergeben werden. Dies ist zwar bei Weitem nicht mit den Möglichkeiten des *PPMs* vergleichbar, stellt aber einen wesentlichen Fortschritt zu der stark aggregierten Sicht dar, die Android dem Nutzer bietet. Zusätzlich lassen sich alle *Permissions* über den *PMP-Gatekeeper* managen, was unter Android ebenfalls nicht der Fall ist. Da auf diese Weise alle Apps kontrolliert werden, ist der Schutz für Nutzer- und Systemdaten vollständig gewährleistet und Erweiterungen können ohne Risiko auch auf Anwendungsebene nachinstalliert werden. Das einzige Manko bei

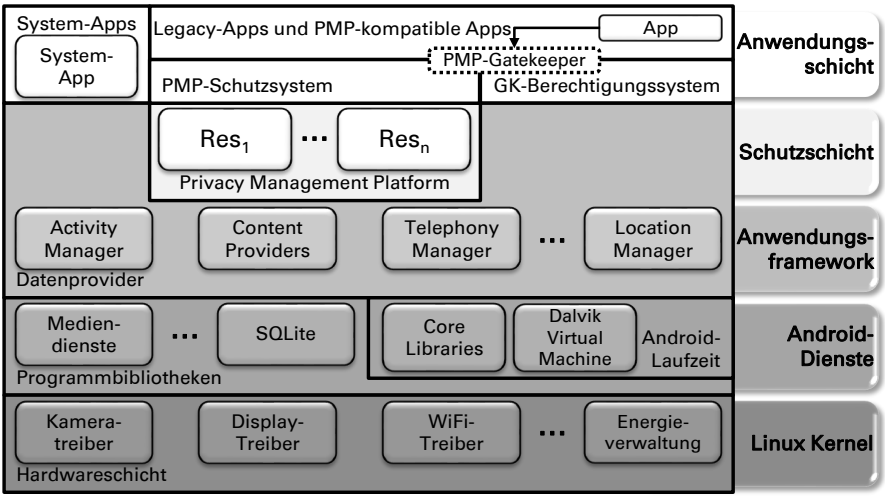


Abbildung 4.26: Eingliederung von *PMP* und *PMP-Gatekeeper* in die Android-Systemarchitektur

diesem Ansatz bleibt die Manipulation an der mobilen Plattform zur Integration der *PMP*. Diese Manipulation fällt allerdings minimal aus und führt zu keinen bemerkenswerten Laufzeitveränderungen.

Abbildung 4.26 zeigt die vollständige Android-Systemarchitektur, die um eine von der *PMP* eingeführte logische Schutzschicht erweitert wurde. Entsprechend dieser Systemarchitektur wurde im Rahmen der Bachelorarbeit von Salsa [Sal14] die *PMP* und der *PMP-Gatekeeper* realisiert.

4.5 Verwandte Arbeiten

Die in diesem Abschnitt diskutierten verwandten Arbeiten befassen sich ausschließlich mit Datenschutzsystemen gemäß der in Abschnitt 1.2 gegebenen Definition. Eine Diskussion verwandter Arbeiten im weiteren Sinne, die sich beispielsweise über Vertrauensmetriken oder Nutzerempfehlungen mit Datenschutzaspekten befassen, findet sich in Abschnitt 3.3. Ein direkter Vergleich eines solchen Ansatzes mit der *PMP* ist nicht zielführend, da dabei zwei unterschiedliche Intentionen anvisiert werden.

Für die Abgrenzung der *PMP* zu den verwandten Arbeiten wird berücksichtigt, inwiefern diese die in Abschnitt 4.2 aufgestellten Eigenschaften realisieren, die für ein Schutzsystem relevant sind. Hierzu gehört, ob sie den Nutzer mit **Feedback** versorgen können, ob **Regeländerungen** auch zur Laufzeit möglich sind, d. h. ob Berechtigungen dynamisch vergeben bzw. entzogen werden können, ob eine **Anonymisierung** der Nutzerdaten mit diesen möglich ist, ob eine **Absturzsicherheit** für Apps gewährleistet wird, auch wenn dieser Berechtigungen entzogen wurden, und ob eine **Kontextsensitivität** für die Berechtigungsregeln gegeben ist, d. h. ob deren Anwendung auch von der aktuellen Situation abhängt. Zusätzlich wird betrachtet, nach welcher **Strategie** das jeweilige Schutzsystem realisiert wurde (siehe Abschnitt 4.3), d. h. ob es darauf setzt, dass die mobile Plattform manipuliert wird (Abschnitt 3.1 / Abschnitt 4.3.3 und Abschnitt 4.3.4) oder dass die Apps auf das Datenschutzsystem angepasst werden müssen (Abschnitt 3.2 / Abschnitt 4.3.5). Dies hat unmittelbar Einfluss auf die Charakteristiken des jeweiligen Datenschutzsystems, wie die Evaluation in Abschnitt 4.3.6 zeigt. Aufgrund der sehr großen Anzahl an Arbeiten, die mit der *PMP* verwandt sind, kann im Folgenden nur ein repräsentativer Auszug aus diesen Arbeiten besprochen werden. Die verwandten Arbeiten werden in alphabetischer Reihenfolge besprochen; aus dieser lässt sich daher keine Wertung ablesen.

Nauman, Khan und Zhang [NKZ10] stellen ein erweitertes Modell für die Vergabe von Android-*Permissions* vor. In diesem Modell können mit jeder *Permission*, die eine App erhält, Restriktion verbunden sein. Beispielsweise kann einer App die `SEND_SMS`-*Permission*, also die Berechtigung SMS-Nachrichten zu verschicken, erteilt werden, jedoch unter der Auflage, dass sie von dieser Berechtigung je Tag höchstens fünfmal Gebrauch machen darf. Wie dieses Beispiel zeigt, kann eine Restriktion mit dem Zustand einer App oder eines Sensors verknüpft sein. Dadurch lassen sich auch Restriktionen formulieren, die auf die Situation, in der eine App verwendet wird, reagieren können. Zur Realisierung dieses Modells führen sie mit *Apex* ein neues Datenschutzsystem ein, das unmittelbar in Android integriert wird. Mit *Apex* wird ein erweiterter Installationsdialog mitgeliefert, über den der Nutzer für jede Android-*Permission* wählen kann, ob er diese einer App gewähren oder verweigern will und ob an den Gebrauch Restriktionen gebunden sind. Über den gleichen Dialog kön-

nen die Berechtigungen einer App auch zur Laufzeit geändert werden. *Apex* beinhaltet auch Komponenten, die die an eine Berechtigung gebundenen Restriktionen auswertet. Für die Vollstreckung der Berechtigungsregeln greift *Apex* allerdings auf die Android-Berechtigungsprüfung zurück. D. h. wurde einer App eine *Permission* entzogen oder hat eine App die für sie definierten Restriktionen für den Gebrauch der *Permission* überschritten, so wird eine Sicherheits-Exception ausgelöst, was in der Regel den Absturz der betroffenen App zur Folge hat. Einen Grund dafür wird dem Nutzer nicht mitgeteilt. Auch lassen sich mit *Apex* keine veränderten, gefilterten oder anonymisierten Werte an eine App weitergeben. Da das zum Einsatz kommende Modell für die Berechtigungsvergabe vollständig auf den Android-*Permissions* basiert, ist es zwar zu allen bestehenden Apps kompatibel, jedoch ist es damit nicht möglich, die bestehenden hochgradig grobgranularen *Permissions* zu verfeinern, um dem Nutzer eine bessere Kontrolle über seine Daten zu geben.

Hornyack et al. [HHJ+11] analysieren mithilfe des Analysewerkzeugs *TaintDroid* [EGC+10] den Informationsfluss von Apps, die sowohl Zugriff auf sensible Nutzerdaten[28] haben als auch die Möglichkeit besitzen, auf das Internet zuzugreifen. Erst durch die Kombination von beidem entsteht nach Ansicht der Autoren eine potentielle Gefährdung, da auf diese Weise private Daten an Dritte weitergegeben werden können. Die Analysen zeigen, dass eine Vielzahl an Apps ihre Berechtigungen dazu missbraucht, um Nutzerdaten an Werbeserver zu schicken. Daher fokussiert sich ihr Schutzsystem *AppFence* auf die unkontrollierte Weitergabe von Daten an externe Informationssenken. Zu diesem Zweck modifiziert *AppFence* das Android Anwendungsframework um zwei Funktionen. Einerseits gestattet es sensible Daten so zu verändern, bevor diese an eine App weitergegeben werden, dass diese keine vertraulichen Informationen mehr preisgeben. Hierfür können leere, gefälschte oder konstante, vom Nutzer vorgegebene Daten genutzt werden. Andererseits kann es das System in den Flugzeugmodus versetzen, wenn eine bestimmte App ausgeführt wird, um zu verhindern, dass diese Daten weiterleiten kann. Apps behalten auf diese Weise ihre Berechtigungen und lösen folglich keine Sicherheits-Exception beim Zugriff auf geschützte Daten aus. Somit ist die Funktionalität der App ggf. redu-

[28] Hornyack et al. identifizieren auf einem Smartphone 12 unterschiedliche Quellen für sensible Daten, z. B. die Kamera, das Mikrofon oder den Kalender.

ziert, aber die Absturzsicherheit bleibt gewahrt. Die Konfiguration von *AppFence* kann zur Laufzeit erfolgen. Ein Feedback für den Nutzer, welche Auswirkungen die von ihm definierten Berechtigungsregeln haben ist in *AppFence* genauso wenig vorgesehen, wie kontextsensitive Regeln. Ein großes Problem bei diesem Ansatz ist, dass sogenannten *Privilege Escalation Attacks*[29] Tür und Tor geöffnet sind, da Apps von *AppFence* nur berücksichtigt werden, wenn sie sowohl einen Internetzugang haben als auch auf sensible Daten zugreifen können. Werden diese Berechtigungen bewusst auf mehrere Apps verteilt, bemerkt *AppFence* dies nicht.

Backes et al. [BGH+12] führen mit *AppGuard* ein Datenschutzsystem ein, das in Apps eine Monitoring-Komponente einbaut, die Apps von innen heraus überwacht. Damit sind keine Manipulationen an der mobilen Plattform erforderlich und Nutzer können das Datenschutzsystem wie eine normale App bei sich installieren. *AppGuard* besteht aus drei Bestandteilen: (1) einem vorkonfiguriertem Regelwerk, dessen Regeln unmittelbar auf Android-*Permissions* abgebildet sind, (2) einem App-Konverter, der die Monitoring-Komponente und das Regelwerk in Apps injizieren kann, und (3) einer GUI, über die der Nutzer weitere, selbstdefinierte Regeln dem Regelwerk hinzufügen kann [BGH+13]. Für die Formulierung der Regeln wird mit *SOSPoX* eine neue, auf *SPoX* [HJ08; HJS12] basierende Sprache eingeführt, mit der sich kontextbasierte Restriktionen für Regeln angeben lassen. Darüber hinaus kann in *SOSPoX* auch beschrieben werden, wie sich der Kontrollfluss einer App ändern soll, wenn ihre reguläre Ausführung gegen das Regelwerk verstoßen würde [BGH+14]. Dank dieser mächtigen formalen Sprache kann der Nutzer exakt definieren, wie Daten einer App zur Verfügung gestellt werden sollen, d. h. ob die Daten unverändert, aggregiert, anonymisiert oder vollständig randomisiert weitergegeben werden sollen. Auch kann er beschreiben, wie eine App in dem Fall reagieren soll, wenn sie aufgrund einer Regel keinen Zugriff auf Daten erhält, wodurch *AppGuard* absturzsicher ist. Trotz der verfügbaren GUI ist die Formulierung einer solchen Regel jedoch hochgradig komplex. Für die meisten Nutzer ist daher das vorkonfigurierte Regelwerk geeignet. Mit diesem können einer App Android-*Permission* zur Laufzeit entzogen werden [BGH+12]. Von dem For-

[29] Bei der *Privilege Escalation Attack*, einer unter Android häufigen Angriffsart, geben Apps Berechtigungen unkontolliert aneinander weiter [DDSW11].

schungsprototyp *AppGuard* gibt es auch eine kommerzielle Version namens *SRT AppGuard Pro*, die von *Backes SRT GmbH* vertrieben wird[30]. Nach einer eingehenden Prüfung durch Google ist *SRT AppGuard Pro* allerdings nach kurzer Zeit aus *Google Play* verbannt worden. Zu diesem Schritt wird aufgrund der liberalen Veröffentlichungspolitik von Google nur selten gegriffen. Allerdings bewegt sich *SRT AppGuard Pro* in einer juristischen Grauzone[31], weshalb dieser Schritt durchgeführt wurde [AP12].

Ringer, Grossman und Roesner [RGR16] schließen eine Manipulation der mobilen Plattform zur Einführung eines neuen Datenschutzsystems kategorisch aus, da diese durch das Android-Fragmentierungsproblem[32] quasi unmöglich gemacht wird. Theoretisch müsste ein solches System für jedes Gerät und für jede Softwareversion speziell angepasst werden. *AUDACIOUS* stellt daher ein Datenschutzsystem dar, das unabhängig von der darunterliegenden Version der mobilen Plattform und der verwendeten Hardware genutzt werden kann. Hierfür wird ein verändertes App-Modell eingeführt, das die Programmbibliotheken von *AUDACIOUS* verwendet. Mit diesen Bibliotheken können Experten sowohl statische als auch dynamische Analysen der App durchführen und diese dadurch sehr effizient vollständig durchleuchten. Jede App definiert in ihrem Programmcode exakt, welche Daten sie wann nutzen möchte und wie diese von ihr verarbeitet werden. Die Experten prüfen in den Analysen, ob diese Datennutzung zu der Funktionalität der App passt. Nur Apps, die in dieser Überprüfung kein auffälliges Verhalten zeigen, dürfen überhaupt veröffentlicht werden. Zur Laufzeit einer App, überwachen diese sich selbst über die eingebauten *AUDACIOUS*-Programmbibliotheken. Sobald sie gegen die definierte Datennutzung verstößt oder aktiv wird, ohne dass dies durch eine Interaktion mit dem Nutzer ausgelöst wurde, bemerkt *AUDACIOUS* dies und stoppt die App. In diesem Ansatz ist nicht vorgesehen, dass der Nutzer mit dem Datenschutzsystem interagiert, sondern dass Experten bewerten, welches App-Verhalten für ihn unproblematisch ist. Daher sind Regeländerungen zur Laufzeit nicht möglich und es lassen sich keine anonymisierten Daten an Apps

[30] siehe http://www.srt-appguard.com

[31] Die Manipulation einer App verstößt gegen §§ 14, 69a und 69c UrhG [BRD16a]. Zusätzlich verstößt die dafür benötigte Dekompilierung der App gegen §§ 202a, 202c Absatz 1 und 303a StGB [BRD16b].

[32] Unter dem Android-Fragmentierungsproblem versteht man, dass Entwickler mit einer großen Anzahl an heterogenen Soft- und Hardwarekonstellationen umgehen müssen [PPH13].

weitergeben, sodass ein Verstoß gegen die definierten Datennutzungsregeln den Absturz der App zur Folge hat. Ringer, Grossman und Roesner gehen davon aus, dass App-Entwickler die nötigen Änderungen an ihren Apps selbst vornehmen. Dennoch wird *AUDACIOUS* im Rahmen der vorliegenden Arbeit zu den App-Konvertern gezählt, auch wenn die Konvertierung manuell erfolgen muss.

Xu, Saıdi und Anderson [XSA12] führen mit *Aurasium* ein Datenschutzsystem ein, mit dem Apps, von denen eine potentielle Bedrohung ausgeht, dahingehend manipuliert werden können, dass der Nutzer deren Zugriffe auf Daten und Funktionen feingranular regulieren kann. Anstelle jedoch den eigentlichen Bytecode einer App zu manipulieren, verwendet *Aurasium* eine zusätzliche Sandbox, in der die App ausgeführt wird. Diese Sandbox wird allerdings nicht auf Systemebene installiert, was eine Manipulation der mobilen Plattform zur Folge hätte, sondern wird in der APK-Datei jeder App injiziert. Xu, Saıdi und Anderson schlagen als Strategie hierfür eine Cloud-basierte Lösung vor, die logisch zwischen den *App-Stores* und den Nutzern angesiedelt ist und die in jede App, die aus einem *App-Store* heruntergeladen wird, automatisch die *Aurasium*-Sandbox einbaut. In der Sandbox befinden sich Monitoring-Komponenten, die jede Anfrage einer App an das Anwendungsframework überwachen und ggf. intervenieren. Dadurch ist *Aurasium* nicht auf die Android-*Permissions* angewiesen, und der Nutzer kann daher sehr viel feingranularere Regeln definieren. So ist es beispielsweise möglich einer App den Zugang zum Internet ausschließlich für bestimmte Adressen zu gewähren. Hierzu informiert *Aurasium* den Nutzer jeweils bei der erstmaligen Anfrage einer App an das Anwendungsframework über die Regulierungsoptionen, die ihm für diesen Anfragetyp zur Verfügung stehen. Hierin besteht jedoch auch ein großer Schwachpunkt dieses Ansatzes. Da *Aurasium* weder eine mächtige Regelsprache verwendet noch dynamisch erweiterbar ist, hängen die verfügbaren Optionen stark vom Anfragetyp ab. Während beispielsweise die Verwendung des Internets somit gut reguliert werden kann, beschränkt sich *Aurasium* bei anderen Anfragetypen darauf, die jeweilige Anfrage zu genehmigen oder abzubrechen. Eine Anonymisierung oder zumindest die Rückgabe von leeren Daten ist nicht vorgesehen, weshalb es bei Apps zu Abstürzen kommen kann, wenn eine Anfrage abgebrochen wird. Auch kann mit den Berechtigungsregeln kein Kontextbezug verknüpft werden.

Conti, Nguyen und Crispo [CNC11] stellen mit *CRêPE* ein kontextsensitives Datenschutzsystem für Android vor. In *CRêPE* kommt ein eigenes, mächtiges System zur Situationserkennung zum Einsatz. Hierfür führen Conti, Zachia-Zlatea und Crispo [CZC11] ein biometrisches Maß ein, das über Sensoren, die in praktisch jedem Smart Device verfügbar sind (z. B. den Beschleunigungssensor oder den Lagesensor), die Bewegungsmuster eines Nutzers eindeutig erkennt. Dadurch kann *CRêPE* Rückschlüsse auf dessen gegenwärtige Aktivitäten ziehen. Mithilfe dieses Maßes können mit *CRêPE* auch höherwertige Kontexte beschrieben werden, die weit über die reine Verknüpfung von Sensorwerten hinausgeht. Jede Berechtigungsregel setzt sich in *CRêPE* stets aus einem Tripel *Subjekt-Objekt-Berechtigung* zusammen. Ein *Subjekt* kann sowohl eine App als auch ein Nutzer[33] sein. Ein *Objekt* kann jede Art von Datenquelle sein. Die *Berechtigung* gibt lediglich an, ob ein *Subjekt* auf ein *Objekt* zugreifen darf. Zu jedem Tripel wird stets ein Kontext, unter dem diese Regel gelten soll, angegeben. Bei der Durchsetzung dieser Regeln verwendet *CRêPE* ein Zugangskontrollsystem auf Basis von eXtensible Access Control Markup Language (XACML) [MCSB08]. Dieses Kontrollsystem wird direkt in das Anwendungsframework von Android eingebaut [CCFZ12]. Neben dieser starken Fokussierung auf kontextsensitive Berechtigungsregeln unterscheidet sich *CRêPE* von vielen anderen Datenschutzsystemen dadurch, dass das Regelwerk zur Laufzeit auch von externen Quellen konfiguriert werden kann. Genau wie die *Presets* in der *PMP* kann dies einerseits dazu genutzt werden, um unerfahrenen Nutzern Anhaltspunkte für eine sinnvolle Regelbasis zu geben und andererseits, um Administratoren eine Möglichkeit zu geben, Regeln zu definieren und an alle Angestellte zu verteilen, die ausschließlich während der Arbeitszeit gelten. Damit kann *CRêPE* als Enabler für BYOD verstanden werden. Der Nutzer selbst steht allerdings weniger im Fokus, weshalb ein Feedback zu Berechtigungsanfragen oder Regeländerungen nicht vorgesehen ist. Auch können Apps nicht mit verfälschten Informationen versorgt werden, wodurch diese abstürzen können, wenn eine Berechtigungsregel ihre Datenzugriffe einschränkt.

Saracino et al. [SMAD16] sehen in der großen Menge an Daten, die von Apps ohne das Wissen des Nutzers beispielsweise an Werbeserver gesendet werden, neben dem reinen Problem des Datenschutzes auch wegen der Kos-

[33] Nutzer werden über ihr eindeutiges biometrische Maß identifiziert.

ten, die mit dem hohen Datenaufkommen verbunden sind, eine Bedrohung für den Nutzer. Daher adressieren sie diese Problematik in *Data-Sluice*. In ihrem Ansatz ist die einzige Ressource, die der Nutzer reglementieren kann, daher der Netzwerkzugriff. Unter Android hat eine App vier unterschiedliche Möglichkeiten, um Daten an das Netzwerk zu senden, angefangen bei sehr hardwarenahen Methoden, wie z. B. die `connect`-Methode eines `Sockets` auf Ebene des Kernels, bis hin zu höherwertigen Methoden, wie z. B. die `loadUrl`-Methode eines `WebViews`, die die empfangenen Daten direkt visualisiert. Die Android-App überwacht daher das System und wird aktiv, sobald eine dieser vier Methoden aufgerufen wird. Dies kann eine von vier Reaktionen nach sich ziehen: (1) *Data-Sluice* erlaubt den Netzwerkzugriff. (2) *Data-Sluice* erlaubt den Netzwerkzugriff, logt ihn aber mit. (3) *Data-Sluice* erlaubt den Netzwerkzugriff nur, wenn das Ziel nicht auf einer Blacklist steht. (4) *Data-Sluice* erlaubt den Netzwerkzugriff nicht. Über das Log (in Regel (2)) erhält der Nutzer ein Feedback zu den Aktionen der App. Selbst wenn eine App keinen Netzwerkzugriff erhält, so stürzt diese in der Regel nicht ab, da dies beispielsweise auch im Flugzeugmodus der Fall sein kann und Apps daher berücksichtigen, dass ein solches Problem auftreten kann. Die Konfiguration, bei welcher App welche Regel angewandt werden soll, erfolgt zur Laufzeit über eine GUI von *Data-Sluice*. Eine solche Überwachung und Manipulation über die Grenzen der eigenen Sandbox hinaus ist jedoch unter Android nicht möglich. Daher greift *Data-Sluice* auf das *Xposed Framework*[34] zurück. Dieses Framework hängt sich direkt in das Android-System ein und stellt anderen Apps Werkzeuge zur Analyse oder Manipulation von Apps oder des Systems zur Verfügung. Zur Nutzung des *Xposed Frameworks* sind allerdings *Root*-Rechte, d. h. eine Manipulation des Systems nötig. Die Anonymisierung von Daten ist aufgrund der Funktionsweise von *Data-Sluice* nicht erforderlich. Auch kontextsensitive Regeln werden nicht unterstützt.

Jeon et al. [JMV+12] adressieren mit ihrer Arbeit das Problem, dass viele App-Entwickler mit den *Permissions* überfordert sind und daher unabsichtlich ihren Apps zu viele Berechtigungen geben [KKKH15]. Damit die Entwickler in der Lage sind, die Berechtigungen besser zu verteilen, spalten Jeon et al. die existierenden *Permissions* in feingranulare Berechtigungen auf. Die neuen Be-

[34] siehe http://repo.xposed.info/module/de.robv.android.xposed.installer

rechtigungen orientieren sich dabei an den häufigsten Aktivitäten von Apps, für die Nutzer- oder Systemdaten nötig sind. Diese Aktivitäten werden in vier Kategorien eingeteilt. Für jede Kategorie werden generische Anonymisierungs- und Filtertechniken definiert, die der Nutzer verwenden kann, um den Datenzugriff von Apps einzuschränken. Die Berechtigungen werden im Datenschutzsystem *Dr. Android & Mr. Hide* realisiert. Zur Durchsetzung der Berechtigungseinstellungen wird mit *hidelib* eine Programmbibliothek zur Verfügung gestellt, die die APIs des Anwandungsframeworks auf Basis der neuen Berechtigungen reimplementiert. Diese Bibliothek wird allerdings nicht in die mobile Plattform eingebracht, sondern von einem Android-Service, d. h. eine App ohne GUI, namens *Mr. Hide* angeboten. Dieser Service wird beim Systemstart gestartet und läuft dauerhaft im Hintergrund. In *Mr. Hide* wird auch das Regelwerk des Systems verwaltet. Damit Apps mit *Mr. Hide* zusammenarbeiten können, kommt ein Werkzeug namens *Dr. Android* zum Einsatz. *Dr. Android* modifiziert den Bytecode einer App dahingehend, dass sämtliche Aufrufe an die Komponenten des Anwendungsframeworks durch gleichwertige Aufrufe an *Mr. Hide* ersetzt werden. Dank der für die Berechtigungskategorien vorhandenen Anonymisierungstechniken erhalten Apps für jede Anfrage Daten zurück. Ein App-Absturz ausgelöst durch fehlende Berechtigungen kann daher ausgeschlossen werden. Die Interaktionsmöglichkeiten mit dem Nutzer sind allerdings stark eingeschränkt. So erhält er kein Feedback von *Dr. Android & Mr. Hide* und kann die Berechtigungen einer App zur Laufzeit nicht ändern. Darüber hinaus lassen sich keine kontextsensitiven Berechtigungsregeln definieren.

Zhang et al. [ZWL+16] befassen sich in ihrer Arbeit nicht direkt mit der Problematik der Berechtigungsvergabe, sondern mit einer damit verwandten Thematik. Da Android Apps weitreichende Möglichkeiten zur Realisierung von IPC über die Grenzen einer Sandbox hinaus bietet (siehe Abschnitt 2.3), kommt es häufig zu einem unkontrollierten (und nicht selten auch ungewollten) Austausch von Daten [GZWJ12] und sogar Berechtigungen [SBG+13] zwischen Apps. Daher führen Zhang et al. mit ihrem *IacDroid* genannten Schutzsystem zwei Komponenten ein, die den Datenaustausch zur Laufzeit überwachen und ggf. reglementieren und regulieren. Die erste Komponente erweitert zu diesem Zweck die Android-*Binder*, die zur Durchführung der IPC benötigt werden, um Monitoring-Funktionalitäten. Mithilfe dieser Komponente zeichnet

IacDroid auf, sobald eine App eine andere App aufruft. Aufgrund der transitiven Eigenschaft dieser Aufrufbeziehung kann *IacDroid* sehr einfach komplexe Aufrufketten ermitteln. Die zweite Komponente erweitert das Android-*Permission*-Kontrollsystem. Wann immer eine App eine Methode aufruft, für die eine spezielle Berechtigung benötigt wird, fängt *IacDroid* die Berechtigungsanfrage ab und informiert den Nutzer darüber, auf welche Daten eine App zugreift und mit welchen Apps sie anschließend kommuniziert, d. h. mit welchen Apps sie möglicherweise diese Daten austauscht. Der Nutzer kann daraufhin entscheiden, ob der Methodenaufruf ausgeführt werden soll. *IacDroid* speichert die Entscheidung ab; bei einem erneuten vergleichbaren Methodenaufruf trifft *IacDroid* selbstständig basierend auf den vorherigen Nutzerentscheidungen eine Entscheidung. Da hierfür das Android-*Permission*-System verwendet wird, sorgt ein Berechtigungsentzug für eine Sicherheits-Exception in der App und damit dafür, dass diese abstürzt. Außerdem ist es nicht möglich, den Kontext bei den Berechtigungsregeln zu berücksichtigen[35].

Beresford et al. [BRSS11] stellen mit *MockDroid* eine reine Erweiterung des Android-Anwendungsframeworks vor. Für ausgewählte Datenquellen (u. a. den Standort oder das Adressbuch) und Systemfunktionen (u. a. Zugriffe auf das Internet oder die Nutzung von SMS-Nachrichten) bietet *MockDroid* erweiterte Schnittstellen an. Mit diesen Schnittstellen können Apps selbstständig die von ihnen benötigten Berechtigungen reduzieren, beispielsweise indem sie nur einen Internetzugang zu einer bestimmten Adresse anfordern. Hierüber sind sie in der Lage, den Nutzer genauer darüber zu informieren, wozu bestimmte Berechtigungen genutzt werden und somit Vertrauen zu schaffen. Zusätzlich kann der Nutzer in *MockDroid* für alle Daten, die über diese Schnittstellen ausgetauscht werden, wählen, ob diese unverändert an die App weitergegeben werden oder ob die App nur vollständig verfälschte Daten erhält. Diese Einstellungen kann er bei Bedarf zur Laufzeit ändern. Im Vergleich zu den Möglichkeiten, die beispielsweise die *PMP* dem Nutzer bietet, ist dies zwar eingeschränkt, dennoch stellt es eine Art Datenanonymisierung dar. Da die App in jedem Fall mit der Art von Daten versorgt wird, die sie erwartet, kommt es durch *MockDroid* zu keinen Sicherheits-Exceptions. Dies alles gilt aber nur für

[35] *IacDroid* bezeichnet sich selbst als ein kontextsensitives Datenschutzsystem. Dies bezieht sich jedoch ausschließlich auf die App-Aufrufketten, in denen ein kritischer Methodenaufruf erfolgt, und nicht auf den Kontext, in dem sich ein Nutzer befindet.

sieben der beinahe einhundertfünfzig *Permissions*, die es aktuell in Android gibt. Daten, auf die eine App mit einer der restlichen *Permissions* zugreifen kann, werden durch *MockDroid* nicht geschützt. Auch können die Berechtigungen einer App nicht mit einem bestimmten Kontext verknüpft sein. Obwohl App-Entwickler durch die teilweise feingranularen Berechtigungen in *MockDroid* dem Nutzer eine Art Rückmeldung darüber geben können, was mit seinen Daten geschieht, gibt ihm das Schutzsystem selbst keinerlei Feedback.

Eines der wenigen Datenschutzsysteme, die weder Apps noch die mobile Plattform manipulieren, stellt *Privacy Protector (No root)*[36] dar. Dieses System sieht ausschließlich die aktuellen Standortdaten sowie den Zugriff auf das Internet als potentielle Gefahrenquellen an. Der Nutzer kann in *Privacy Protector (No root)* festlegen, welche Apps keinen Zugriff auf seinen Standort haben sollen oder das Internet nutzen dürfen. Hierzu scannt *Privacy Protector (No root)* dauerhaft, welche Apps aktuell auf dem Smart Device ausgeführt werden. Zu diesem Zweck stellt der Android-Activity-Manager die Methode `getRunningTasks` zur Verfügung. Diese Methode ermittelt alle Tasks, die aktuell aktiv oder pausiert sind. Stellt die *Privacy-Protector-(No root)*-App fest, dass sich eine App in dieser Liste befindet, deren Berechtigungen vom Nutzer eingeschränkt wurden, dann deaktiviert sie das GPS oder sie versetzt das System in den Flugmodus. So kann verhindert werden, dass Apps aufgrund eines Berechtigungsentzugs abstürzen. Von diesen Einschränkungen sind allerdings automatisch alle anderen Apps, die zum gleichen Zeitpunkt laufen, ebenfalls betroffen. Auch wenn ein Nutzer die Regeln in *Privacy Protector (No root)* zur Laufzeit ändern kann, hat dies daher keinen unmittelbaren Einfluss auf die Berechtigungen einer einzelnen App; es gilt stets die Summe aller Restriktionen der gleichzeitig ausgeführten Apps. Ein weiteres großes Problem dieses Ansatzes ist, dass die `getRunningTasks`-Methode seit API-Level 21, d. h. mit der Android-Version 5.0, stark eingeschränkt wurde, um Apps die Möglichkeit zu nehmen, darüber das Nutzerverhalten auszuspionieren. Daher werden die meisten Apps aus der Ergebnismenge gefiltert, wodurch *Privacy-Protector-(No root)* nicht mehr zuverlässig feststellen kann, ob eine bestimmte App gerade ausgeführt wird. Der Nutzer erhält bei diesem Ansatz keinerlei Feedback

[36] siehe https://play.google.com/store/apps/details?id=net.houzuo.android.privacyprotector

und kann seine Daten nicht anonymisieren oder Kontextbedingungen an die Berechtigungen einer App knüpfen.

Davis et al. [DSKC12] beschreiben einen Ansatz, dessen Ziel es ist, dem Nutzer die maximale Kontrolle über seine Daten zu geben. Mit dem generischen App-Konverter *I-ARM Droid* kann ein Nutzer beliebige Android-Methoden angeben, die er als kritisch erachtet. Dabei kann es sich um jede Art von Methodenaufruf handeln, auch solche, für die unter Android keinerlei *Permissions* erforderlich sind. Zu jedem dieser Aufrufe muss der Nutzer einen alternativen Aufruf oder ein Code-Fragment angeben, das statt dieses Aufrufs ausgeführt werden soll. Der App-Konverter manipuliert daraufhin auf Bytecode-Ebene eine App und fügt in diese Monitoring-Komponenten für die in den Regeln angegebenen Methoden ein. Die Monitoring-Komponenten führen dann zur Laufzeit die vom Nutzer definierten Ersetzungen durch, sobald eine dieser Methoden aufgerufen wird. Obwohl die Möglichkeiten, die dem Nutzer dadurch gegeben werden nahezu unbegrenzt sind, ist ein normaler Nutzer mit dieser Aufgabe überfordert. Daher geben Davis und Chen [DC13] in ihrem auf *I-ARM Droid* basierenden Nachfolger *RetroSkeleton* diese Aufgabe an einen Experten ab. Dieser formuliert in *Clojure*[37], welche Methoden oder Methodenfolgen wie zu ersetzen sind, und *RetroSkeleton* realisiert die Umsetzung in einer App. Im Gegensatz zu *I-ARM Droid* kommen allerdings keine Monitoring-Komponenten zum Einsatz, sondern es findet eine echte Ersetzung des Codes statt. Aufgrund der generischen Formulierung der Ersetzungsregeln, sind diese nicht auf eine bestimmte App ausgerichtet, sondern lassen sich auf jede beliebige App anwenden. *Clojure* ist als *Lisp*-Dialekt Turing-vollständig, wodurch in der Theorie dieser Ansatz dem Schutzsystem maximale Freiheiten verleiht. Betrachtet man aber die Eigenschaften, die *I-ARM Droid* sowie *RetroSkeleton* an sich mitbringen, so unterstützen diese kein Feedback, keine Anonymisierung von Daten und auch keine kontextsensitiven Regeln. Dies alles kann erst durch sehr komplexe Ersetzungsregeln in das System eingebracht werden. Regeländerungen zur Laufzeit sind nicht möglich, da hierfür stets eine erneute Konvertierung der App erforderlich ist. Auch die Absturzsicherheit ist nicht gegeben, da die Manipulation von Bytecode auf Methodenebene sehr leicht fehleranfällig ist und unabsehbare Nebeneffekte in einer App auslösen kann. Ein weiteres Problem

[37] http://clojure.org/

besteht darin, dass es häufig nicht nur einen Weg (und damit eine Methode) gibt, um auf ein bestimmtes Datum zuzugreifen. Dies erhöht die Komplexität der Ersetzungsregeln ungemein.

Banuri et al. [BAK+12] untersuchen, inwiefern sich die Reihenfolge, in der Berechtigungen von einer App angefordert werden, auf das Gefährdungspotenzial der App auswirkt. So kann eine App, die erst Zugriff aufs Internet erhält und anschließend auf sensible Daten zugreift, weniger Schaden anrichten, als eine App, die diese Operationen in umgekehrter Reihenfolge ausführt. Banuri et al. definieren daher mehrere Operationsfolgen, die auf einen potentiellen Datenmissbrauch hindeuten. Mit *SEAF* führen sie ein Datenschutzsystem für Android ein, das Apps zur Laufzeit auf diese Operationsfolgen hin untersucht. Wird eine solche Folge festgestellt, informiert *SEAF* den Nutzer und er kann entscheiden, ob die Operationsfolge ausgeführt werden darf oder ob eine der dafür benötigten *Permissions* verweigert werden soll. Zur Realisierung von *SEAF* wird der Android-Berechtigungsprozess im Paketmanager abgeändert (siehe Abbildung 3.1). In dieser abgeänderten Version des Paketmanagers wird die Sequenz der angeforderten *Permissions* abgespeichert. Solange diese Sequenz unkritisch ist, verhält sich *SEAF* genau wie das Android-Berechtigungssystem. Erst wenn ein potentiell gefährliches Muster entdeckt wird, wird eine Interaktion mit dem Nutzer erforderlich. Dieser erhält eine Erklärung, welcher Schaden durch das Zugriffsmuster ausgehen kann und kann entscheiden, ob er der App vertraut bzw. ob das Zugriffsmuster für die Funktionalität der App erforderlich ist. Um den Nutzer nicht unnötig mit Berechtigungsanfragen zu belästigen und den Rechen-Overhead für die Überprüfungen gering zu halten, führt *SEAF* ähnlich wie Windows Phone ein Kammermodell ein (siehe Abschnitt 2.2). *System-Apps* und vom Nutzer als sicher eingestufte Apps laufen in einer speziellen Kammer, die von *SEAF* nicht überwacht wird. Jedoch ermöglicht es *SEAF* dem Nutzer nicht, eine App mit anonymisierten Daten zu versorgen, wenn ein kritisches Zugriffsmuster erkannt wurde. Entzieht der Nutzer der App die für das Muster nötigen *Permissions*, so stürzt diese ab. Auch wenn die Berücksichtigung der Funktionsfolge eine Art Ausführungskontext darstellt, ist dies nicht mit der Kontextsensitivität, die beispielsweise die *PMP* ermöglicht, vergleichbar.

Fragkaki et al. [FBJS12] führen ein formales Modell ein, mit dem sowohl Android-*Permissions* als auch die Kommunikation zwischen Apps und deren Aufrufketten abgebildet werden können. Mithilfe dieses Modells kann das Verhalten eines Android-Systems bezüglich des Umgangs mit Berechtigungen beschrieben werden. Insbesondere kann damit, im Gegensatz zu dem Modell, das Android verwendet, Berechtigungen eine Gebrauchsdauer zugeordnet werden, um zu beschreiben, wie lange, d. h. für welche Funktionen eine Berechtigung benötigt wird. Dieses formale Modell kann anschließend automatisch ausgewertet werden, um dabei potentielle Schwachstellen aufzudecken. Während der Datenzugang einer isolierten Android-App durch die *Permissions* eindeutig reglementiert werden kann, ergeben sich insbesondere für Apps, die IPC verwenden, viele Sicherheitslücken. Insbesondere da die Gültigkeit einer *Permission* weder zeitlich noch funktional eingeschränkt werden kann, können diese dadurch leicht von anderen Apps ausgenutzt werden. Mit *Sorbet* führen Fragkaki et al. ein Datenschutzsystem ein, das App-Entwicklern eine Möglichkeit bietet, die ungewollte Weitergabe von *Permissions* zu verhindern und dennoch IPC zu nutzen. Mit dem System kann eine kontrollierte Delegation von Berechtigungen und Daten erfolgen. Hierzu zeichnet *Sorbet* für alle Daten, die zwischen Apps ausgetauscht werden, auf, woher die Daten oder Berechtigungen stammen und an wen sie weitergegeben wurden. Jedes Datum und jede Berechtigung besitzt dabei die Gültigkeitsdauer, die der Datenerzeuger, d. h. die App, die sie erstmals über das Anwendungsframework erhalten oder generiert hat, für sie festgelegt hat. Für jede Weitergabe wird ausgewertet, ob diese den definierten Delegationsregeln entsprechen. Für Aufrufketten werden dabei die Regeln aller Apps in dieser Kette berücksichtigt. Spricht eine Regel gegen die Weitergabe oder ist die Gültigkeitsdauer abgelaufen, so verhindert *Sorbet* diese. Hierzu wird es direkt in den Activity-Manager der mobilen Plattform eingebunden. Da *Sorbet* auf die Entwickler von Apps ausgelegt ist und diesen einen Schutz vor einem Missbrauch ihrer Apps bieten soll, bietet es dem Nutzer keine Interaktionsmöglichkeiten. Die von den Entwicklern vorgegebenen Regeln können nicht verändert werden, Daten können nicht anonymisiert werden und es kann kein Kontextbezug hergestellt werden[38]. Die Absturzsicherheit

[38] *Sorbet* bezeichnet sich selbst als ein kontextsensitives Datenschutzsystem. Dies bezieht sich jedoch ausschließlich auf die App-Aufrufketten, in denen ein kritischer Methodenaufruf erfolgt, und nicht auf den Kontext, in dem sich ein Nutzer befindet.

ist dennoch gewährleistet, da Abbrüche von IPC vorkommen können und ein solcher daher von Apps in der Regel behandelt wird.

Russello et al. [RCFZ11] führen ein feingranulares Modell für Berechtigungen ein, um dem Problem der unbeschränkten Weitergabe von Daten aus Nutzersicht Herr zu werden. Daher besitzt der Nutzer in diesem Modell die Möglichkeit, seine Daten mit Kennzeichnern zu versehen und so auf Datenebene festzulegen, an welches Ziel diese gesendet werden dürfen. Als Ziel kommen sowohl andere Apps als auch externe Empfänger (z. B. Internetserver) in Frage. Mit dem Datenschutzsystem *YAASE* wird die Durchsetzung dieser Regeln realisiert. Hierzu wird auf *TaintDroid* [EGC+10] zurückgegriffen. Mit *TaintDroid* lässt sich der Informationsfluss sowohl innerhalb einer App als auch zwischen mehreren Apps überwachen. *YAASE* nutzt diese Funktionalität dazu, die vom Nutzer gekennzeichneten Daten zu verfolgen. Um auch die Kommunikation mit Internetservern überwachen zu können, modifiziert *YAASE* zusätzlich auf Kernel-Ebene die zum Verbindungsaufbau nötigen Methoden. Dadurch wird *YAASE* immer informiert, wenn eine Verbindung aufgebaut wird und was das Ziel dieser Verbindung ist. *TaintDroid* kann über den App-internen Informationsfluss feststellen, welche Daten an diese Verbindung gesendet werden sollen. Sobald eine Datenweitergabe an eine App via IPC oder an einen Internetserver via `Sockets` festgestellt wird, die den Berechtigungen widerspricht, greift *YAASE* ein. Der Nutzer kann festlegen, in welcher Form dies erfolgen soll, z. B. indem die Daten verfremdet werden. Für die Realisierung dieser Funktionalitäten sind allerdings weitreichende Manipulationen an der mobilen Plattform nötig. In *YAASE* liegt der Fokus auf einer handhabbaren Sprache zur Formulierung der Berechtigungen. Für den Nutzer wird zwar eine GUI zur Verfügung gestellt, über die er zur Laufzeit die Regeln ändern kann, aber auf ein Feedback wird verzichtet. Dadurch dass der Nutzer die Möglichkeit hat, seine Daten zu verfremden, bekommen Apps Änderungen an den Berechtigungen nicht mit und stürzen auch daher nicht ab. Kontextsensitive Berechtigungsregeln lassen sich in *YAASE* allerdings nicht formulieren.

Tabelle 4.2 bietet einen Überblick über die betrachteten verwandten Arbeiten. Als Referenzwerte sind zusätzlich das in Android eingesetzte *Permission*-System sowie die *PMP* aufgeführt.

	Feature					
System	Feedback	Regeländerungen	Anonymisierung	Absturzsicherheit	Kontextsensitivität	**Strategie**
Android	✗	✗	✗	(✓)	✗	Berechtigungen
Apex	✗	✓	✗	✗	✓	System
AppFence	✗	✓	✓	✓	✗	System
AppGuard	✗	✓	✓	✓	✓	Konverter
AUDACIOUS	✗	✗	✗	✗	✓	Konverter*
Aurasium	✓	✓	✗	✗	✗	Konverter
CRêPE	✗	✓	✗	✗	✓	System
Data-Sluice	✓	✓	✗	✓	✗	System
Dr. Android & Mr. Hide	✗	✗	✓	✓	✗	Konverter
IacDroid	✓	✓	✗	✗	✗	System
MockDroid	✗	✓	✓	✓	✗	System
Privacy Protector	✗	✗	✗	✓	✗	App
RetroSkeleton	(✓)	✗	(✓)	✗	(✓)	Konverter
SEAF	✓	✓	✗	✗	(✓)	System
Sorbet	✗	✗	✗	✓	✗	System
YAASE	✗	✓	✓	✓	✗	System
PMP	✓	✓	✓	✓	✓	*PMP-Gatekeeper*

* AUDACIOUS setzt voraus, dass spezielle Schutzbibliotheken in Apps genutzt werden.

Tabelle 4.2: Vergleich von Schutzsystemen für mobile Plattformen (in Anlehnung an [SM13; Sta13b])

Wie man bei dem Vergleich sehen kann, besitzt keines der aktuell verfügbaren Berechtigungssysteme alle Eigenschaften der *PMP*. Die Evaluation in Abschnitt 7.1 zeigt allerdings, dass diese Eigenschaften wichtig sind, um die in Abschnitt 1.3 aufgestellten Schutzziele [DP1] bis [DP6] zu erfüllen. Daher kann die Kombination dieser Eigenschaften als Alleinstellungsmerkmal der *PMP* angesehen werden.

Um neben den Datenschutz- auch die Datensicherheitsziele erfüllen zu können, wird in Kapitel 5 mit dem *Secure Data Container (SDC)* eine Datensicherheitskomponente eingeführt, die aufgrund der Erweiterbarkeit der *PMP* unmittelbar in diese eingebunden werden kann und sie dadurch zu einem holistischen Sicherheitssystem vervollständigt.

Kapitel 5

Der Secure Data Container

Bei der Konzeption und Implementierung des *Privacy Policy Models (PPMs)* und der *Privacy Management Platform (PMP)* wurde darauf geachtet, dass diese allen Anforderungen an ein Datenschutzsystem gemäß der Definition in Abschnitt 1.2.2 genügen und dadurch die in Abschnitt 1.3 definierten Schutzziele erfüllen. Da Apps jedoch häufig dazu neigen, übermäßig viele Daten zu sammeln und diese auch abzuspeichern [LEPM12], wodurch auf einem Smart Device Daten aus den unterschiedlichsten Domänen angehäuft und verknüpft werden [GSG14], rücken diese Geräte immer mehr in den Fokus von Datendieben [Tej13]. Aus diesem Grund reicht ein reines Datenschutzsystem daher nicht aus, um die Daten eines Nutzers angemessen zu schützen. Vielmehr wird zusätzlich ein Datensicherheitssystem benötigt, das die gespeicherten Daten gegen illegale Zugriffe absichert.

Im Folgenden wird daher mit dem *Secure Data Container (SDC)* ein Datensicherheitssystem zur sicheren Speicherung sensibler Daten eingeführt. Neben der reinen Speicherung von Daten führt der *SDC* zusätzlich einen einfachen, aber dennoch sicheren Mechanismus zum Austausch von Daten zwischen Apps ein. Hierbei werden explizit die diesbezüglichen Schwachstellen der vorherrschenden mobilen Plattformen adressiert (siehe Kapitel 2). In diesen können Apps ihre Daten entweder, wie im Fall von iOS, nur sehr eingeschränkt unterein-

ander austauschen oder der Austausch findet quasi unkontrolliert statt, wie es in Android der Fall ist [CFGW11; YY12]. Im Gegensatz dazu wird im *SDC* durch eine vollständige Verschlüsselung aller gespeicherter Daten sichergestellt, dass nur solche Apps Zugriff auf die Daten erhalten, die zuvor vom Nutzer dazu legitimiert wurden. In einem zweiten Schritt wird der *SDC* dahingehend erweitert, dass er zusätzlich einen Datenaustausch zwischen Nutzern ermöglicht. Eine solche Funktionalität ist bisher in keiner der vorherrschenden mobilen Plattformen vorgesehen. Darüber hinaus besitzt der *SDC* weitere Sicherheitsfeatures, wie beispielsweise einen Mechanismus, mit dem Daten sicher und unwiederbringlich gelöscht werden können.

Der *SDC* ist als *Resource* für die *PMP* konzipiert und realisiert. Da der *SDC* alle Anforderungen an ein Datensicherheitssystem erfüllt, wird die *PMP* dadurch zu einem holistischen Sicherheitssystem erweitert. Dennoch sind die Konzepte und Modelle plattformunabhängig und lassen sich auch auf andere Systeme anwenden.

In Abschnitt 5.1 wird zunächst ein einfacher Datencontainer als Erweiterung für die *PMP* vorgestellt, der einen sicheren und regulierbaren Datenaustausch zwischen Apps ermöglicht. Dieser Datencontainer besitzt noch keinerlei Merkmale eines Datensicherheitssystems und dient lediglich als Basis für den *SDC*. In Abschnitt 5.2 werden dessen spezielle Eigenschaften vorgestellt und Abschnitt 5.3 beschreibt dessen Funktionsweise. Abschnitt 5.4 stellt Erweiterungen für den *SDC* vor, die einer verbesserten und überarbeiteten Version des *SDCs* dem sogenannten *Secure Data Container Plus (SDC+)*, Einzug halten. U. a. führt der *SDC+* einen Mehrbenutzerbetrieb mit unterschiedlichen Nutzerkonten ein. Dem *SDC+* werden abschließend in Abschnitt 5.5 verwandte Arbeiten, d. h. Datensicherheitssysteme für mobile Plattformen, gegenübergestellt.

Der *SDC* wurde im Rahmen der eigenen Publikation [SM15] konzipiert, implementiert und vorgestellt. Der *SDC+* wurde in der eigenen Publikation [SM16a] vorgestellt, in deren Rahmen auch dessen Implementierung erfolgte. Der *SDC* fand bei der Realisierung anderer eigener Arbeiten Anwendung, beispielsweise bei *Secure Candy Castle (SCC)* [Sta16] (siehe Abschnitt 6.2), in deren Rahmen auch die Funktionsweise des *SDCs* demonstriert wurde.

5.1 Datenaustausch zwischen Apps mittels der PMP

Nachdem durch die *PMP* bereits sichergestellt ist, dass Apps nur auf Daten zugreifen dürfen, für die sie eine Berechtigung besitzen, d. h. die der Nutzer für sie freigegeben hat, soll mit dem *SDC* sichergestellt werden, dass sie auch nicht illegal an diese Daten gelangen. Ohne den *SDC* wäre dies beispielsweise durch die Ausnutzung von Softwarefehlern in der mobilen Plattform oder durch einen unkontrollierten Datenaustausch zwischen Apps möglich.

In einem ersten Schritt wird mit einem Datencontainer für die PMP, dem *PMP-Datencontainer (PDC)*, eine Komponente eingeführt, mit der Datenaustausch zwischen Apps nicht nur wesentlich sicherer, sondern auch erheblich unkomplizierter realisiert werden kann, als dies unter Android der Fall ist. Im Folgenden liegt der Fokus auf Android, da dies eine der wenigen mobilen Plattformen ist, die einen Datenaustausch zwischen Apps unterstützt (siehe Kapitel 2).

Grundsätzlich besitzt unter Android jede App ihren privaten Speicherbereich, auf den andere Apps nicht zugreifen können. In diesem privaten Speicher können Apps auch *SQLite*-Datenbanken[1] erzeugen. Hierfür ist es allerdings erforderlich, dass ein App-Entwickler die Datenbank jedes mal von Grund auf neu aufsetzt, was sehr viel Programmier-Overhead verursacht[2]. Ein Großteil dieses Codes ist für viele Apps gleich oder zumindest sehr ähnlich. Dennoch ist jeder Entwickler gezwungen, diesen für jede seiner Apps, die eine Datenbank verwendet, erneut zu erstellen.

Da die Datenbank im privaten Speicher einer App liegt, kann diese nur von ihrem Besitzer, d. h. von der App, die sie erzeugt hat, verwendet werden. Damit die darin befindlichen Daten dennoch anderen Apps zur Verfügung gestellt werden können, führt Android einen auf Inter-Process Communication (IPC) basierenden Mechanismus namens *Content Provider* ein. Mit einem *Content Provider* können öffentliche Zugänge zu jeder Form von strukturierten Daten geschaffen werden. Um einen solchen Zugang zu erzeugen, muss der App-Entwickler für seine Datenbank einen *Content Provider* implementieren, in dem eine Uniform Resource Identifier (URI) definiert ist. Über diese URI können Apps, die in anderen Prozessen ausgeführt werden, auf den *Content Provider*

[1] siehe https://www.sqlite.org/
[2] siehe Android Developers [And16i]

und damit auf die Daten in der Datenbank zugreifen. Nach außen hin stellt ein *Content Provider* eine Schnittstelle zur Verfügung, über die Apps Anfragen in einer an Structured Query Language (SQL) angelehnten Sprache stellen können. Auf der Ergebnismenge existiert ein Cursor[3], über den die anfragende App auf die Daten zugreifen kann [And16d]. Für jede Datenquelle, d. h. für jede einzelne Tabelle, in der sich Daten befinden, die mit anderen Apps geteilt werden sollen, wird ein individueller *Content Provider* benötigt. Dadurch steigt der Programmier-Overhead für einen App-Entwickler weiter an.

Das Hauptproblem bei diesem Ansatz liegt allerdings in der Datensicherheit. Der App-Entwickler hat keinen Einfluss darauf, welche Apps auf die Daten seiner App über *Content Provider* zugreifen. Ist einer App die URI des *Content Providers* bekannt, hat sie automatisch vollen und unregulierten Zugriff auf diesen [EOM09]. Zwar kann der App-Entwickler unterschiedliche *Content Provider* für die gleiche Datenbasis angeben (z. B. für einen separierten lesenden und schreibenden Zugriff) oder eigene *Permissions* für den Zugriff definieren [And16e], aber beides hindert eine App nicht wirklich daran einen Datenzugang zu erhalten. Damit andere Apps einen *Content Provider* nutzen können, ist dessen Entwickler gezwungen, sowohl dessen URIs als auch die nötigen *Permissions* bekannt zu geben. Mit diesem öffentlichen Wissen können die Entwickler von Schadsoftware Apps schreiben, die diese URIs ansprechen und alle dafür benötigten *Permissions* anfordern [OMEM09]. Theoretisch können die Nutzer einer solchen App einen Riegel vorschieben, indem sie ihr die *Permissions* verweigern, d. h. die App bei sich nicht installieren. Allerdings zeigen die Untersuchungen von Felt et al. [FHE+12], dass die Nutzer bereits mit den standardisierten Android-*Permissions* überfordert sind – der Bedeutung dieser zusätzlichen *Content-Provider-Permissions* sind sie sich daher auch nicht bewusst.

In Abbildung 5.1 ist der Prozess des Datenzugriffs über einen *Content Provider* in einer an Business Process Model and Notation (BPMN)[4] angelehnten Notation dargestellt. Die Rolle *DB-Besitzer* beschreibt dabei die App, die Daten erzeugt und über *Content Provider* mit anderen teilt. *DB-Nutzer* steht für eine beliebige App, die auf die geteilten Daten zugreift. Zunächst erstellt der

[3] siehe Android Developers [And16f]
[4] siehe http://www.bpmn.org/

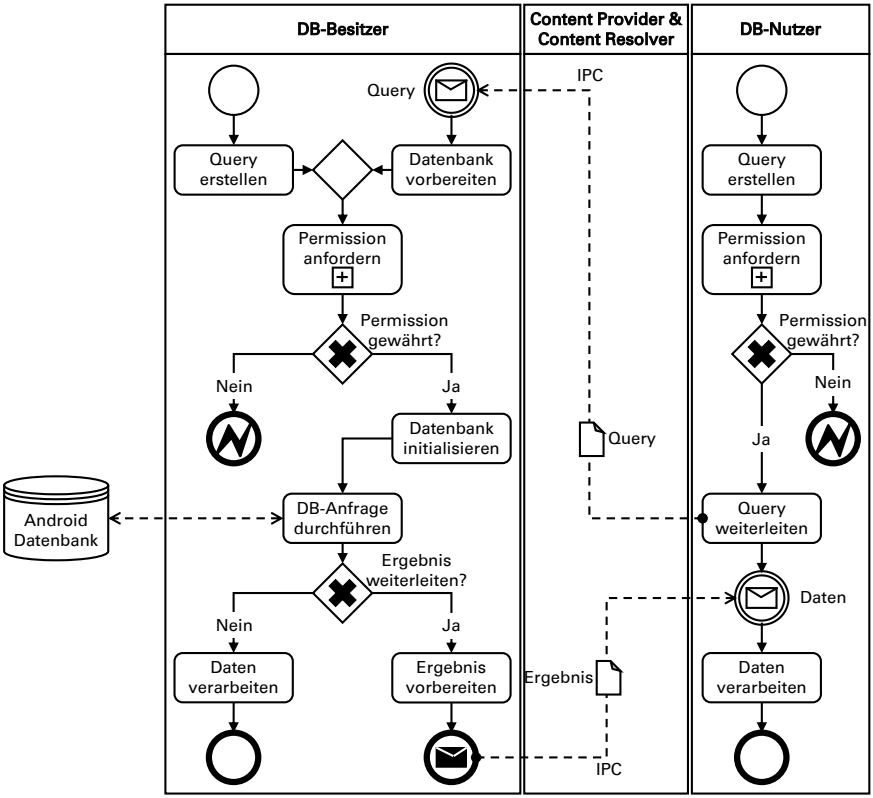


Abbildung 5.1: Prozess des Datenzugriffs über den *Content-Provider*-Mechanismus (in Anlehnung an [SM16a])

DB-Nutzer eine Datenbankanfrage und leitet diese an den *Content Resolver* über dessen `query`-Methode weiter. Der *Content Resolver* ist das Gegenstück zum *Content Provider*, das im Prozess des *DB-Nutzers* ausgeführt wird. Dieser initiiert die IPC mit dem *Content Provider* des *DB-Besitzers* über dessen URI. Ob für diesen Zugriff *Permissions* nötig sind, wird von Android geprüft, sobald der Kommunikationskanal geöffnet wird. Ist dies der Fall, prüft Android, ob der *DB-Nutzer* die nötigen Berechtigungen besitzt. Besitzt er diese nicht, löst Android eine Sicherheits-Exception aus und die *DB-Nutzer*-App stürzt ab. Liegen alle Berechtigungen vor, kann die Kommunikation stattfinden und die Query

wird an den *DB-Besitzer* geschickt. Der *Content Provider* startet daraufhin die *DB-Besitzer*-App und deren Datenbank.

Ab diesem Moment wird die Query verarbeitet, als ob es sich um eine orginäre Anfrage der *DB-Besitzer*-App handeln würde. Es wird von Android geprüft, ob der *Content Provider*, d. h. die *DB-Besitzer*-App, die nötigen *Permissions* besitzt, um die angeforderten Daten zu lesen. Falls nicht, wird auch hier eine Sicherheits-Exception ausgelöst und die *DB-Besitzer*-App stürzt ab. Da das Android-Berechtigungssystem einen Berechtigungsentzug nicht vorsieht, erfolgen beide Überprüfungen jedoch nur pro forma.

Nach der Berechtigungsprüfung wird die Datenbank initialisiert und die Anfrage durchgeführt. Wurde die Anfrage durch einen *Content Provider* initiiert, so fasst dieser die Ergebnismenge in einem `Cursor`-Objekt zusammen und sendet dieses an den *Content Resolver* des anfragenden *DB-Nutzers* zurück. Dieser kann die angefragten Daten daraufhin lokal weiterverarbeiten.

Dieser Ansatz zum Datenaustausch zwischen Apps verursacht einen erheblichen organisatorischen Mehraufwand für App-Entwickler. Diese müssen nicht nur für jede Datenquelle ihrer App, die öffentlich zugänglich sein soll, einen eigenen *Content Provider* erstellen, sondern zusätzlich sicherstellen, dass alle Apps, die diese *Content Provider* nutzen wollen, die zugehörigen URIs kennen. Darüber hinaus funktionieren *DB-Nutzer*-Apps nur so lange auch alle zugehörigen *DB-Besitzer*-Apps auf dem Gerät installiert sind. Da Nutzer sich dieser Wechselwirkung nicht zwangsläufig bewusst sind, bzw. nicht wissen, welche Abhängigkeitsbeziehungen zwischen ihren Apps herrschen, kann dies die Stabilität der mobilen Plattform beeinträchtigen, wenn eine zentrale App gelöscht wird. Hinzu kommt, dass sobald eine App ihre *Content Provider* verändert, alle Apps, die auf die Daten dieser App angewiesen sind, ebenfalls überarbeitet werden müssen.

Die größte Schwachstelle bei diesem Ansatz liegt allerdings in der Performanz und der Datensicherheit respektive dem Datenschutz. Die Performanz wird im Falle eines Datenzugriffs verschlechtert, da die *DB-Besitzer*-App gestartet und im Hintergrund ausgeführt werden muss. Dies kostet zusätzliche Rechenleistung und beeinflusst somit die Akkuentladung negativ. Die Datensicherheit ist gefährdet, da Apps auf diese Weise nahezu unkontrolliert Daten weiterreichen können [EGC+10]. Der Datenschutz ist gefährdet, wenn man

```
1 package de.unistuttgart.ipvs.pmp.resourcegroups.pdc.aidl;
2
3 interface IPDCResource {
4    String get(in String key);
5    String put(in String key, in String value);
6    String delete(in String key);
7 }
```

Quelltext 5.1: Schnittstellendefinition des *PDCs* in AIDL

die *DB-Besitzer*-App als Besitzer der Daten ansieht. Sie hat kein Wissen darüber, welche Apps auf ihre Daten zugreifen und folglich auch keine Möglichkeit dies einzuschränken oder zu verhindern. Beides wird in Abschnitt 7.1 respektive Abschnitt 7.2 detailliert.

Ein *Content Provider* stellt also eine generalisierte Schnittstelle zum strukturierten Zugriff auf Datenquellen, die außerhalb einer App erzeugt und verwaltet werden, dar. Genau dies leisten aber auch *Resources* der *PMP*. Zusätzlich bieten diese Nutzern über die *Privacy Settings* eine feingranulare Regulierung der Zugriffsrechte auf die von ihnen verwalteten Daten. Als ersten Schritt, um sich der Datenschutz- und Datensicherheitsproblematik der *Content Provider* anzunehmen, wird mit dem *PDC* daher ein einfacher generischer Datencontainer als *Resource* eingeführt. Apps können den *PDC* verwenden, um ihre Daten abspeichern und lesen zu können. Da der *PDC* von mehreren Apps genutzt werden kann, stellt der *PDC* ein Möglichkeit dar, um sehr einfach Daten zwischen Apps austauschen zu können.

Zur Realisierung dieser Funktionalitäten erzeugt die *PDC -Resource* eine *SQLite*-Datenbank mit einem generischen *Key-Value*-Datenschema. Aufgrund des generischen Schemas, ist der *PDC* kompatibel zu jeder App und kann jede Art von Daten speichern. Die Schnittstelle des *PDCs* ist an der eines einfachen *Key-Value-Stores* angelehnt. Quelltext 5.1 zeigt die Schnittstelle der *Resource* in Android Interface Definition Language (AIDL).

Mit der `get`-Methode können Apps die Werte, die zu einem gegebenen Schlüssel im *PDC* hinterlegt sind, abrufen. Existiert zu dem Schlüssel kein Eintrag, so wird `null` zurückgegeben. Die `put`-Methode legt den übergebenen Wert unter dem gegebenen Schlüssel ab. Falls zu dem Schlüssel bereits ein

Eintrag existiert, so wird dieser überschrieben. Der vormals gespeicherte Wert wird an den Aufrufer zurückgegeben. Die `delete`-Methode löscht das zu dem übergebenen Schlüssel hinterlegte Schlüssel-Wert-Paar und gibt den gelöschten Wert zurück.

Aufgrund des flexiblen und erweiterbaren Konzepts der *Resource Groups*, können parallel zu diesem *PDC* mit dem generischen *Key-Value*-Datenschema noch weitere Implementierungsvarianten mit individualisierten und spezialisierten Schemata erstellt werden. In diesem Fall sind ebenfalls Anpassungen an der Schnittstelle nötig. Dadurch kann der *PDC* bei Bedarf auf Anwendungsfälle angepasst werden, um die Gebrauchstauglichkeit sowie die Performanz zu erhöhen. Der Nutzer kann über die *PMP* für jede App festlegen, welche Instanz des *PDCs* diese nutzen darf. Hierzu existiert zu jeder der drei Zugriffsarten ein boolesches *Privacy Setting*. Für komplexere Datenschemata und Schnittstellendefinitionen können auch weitere *Privacy Settings* eingeführt werden, um den Datenzugriff besser reglementieren zu können.

Der Prozess des Datenzugriffs über den *PDC* ist in Abbildung 5.2 in einer an BPMN angelehnten Notation dargestellt. Der *PDC* nimmt hierbei die Rolle des *DB-Besitzers* ein. Der *DB-Nutzer*, die anfragende Instanz, ist in zwei Unterinstanzen (App und Nutzer) aufgeteilt. Die *PMP* tritt an die Stelle des Vermittlers zwischen dem *PDC* und der anfragenden Instanz.

Benötigt eine App Zugriff auf die Daten des *PDCs*, so wird die *PMP* automatisch aktiv. Sie prüft, ob die App alle benötigten Berechtigungen besitzt und ob der Nutzer das *Service Feature* aktiviert hat, das die App ausführen möchte. Ist dies nicht der Fall, so gestattet die *PMP* den Zugang nicht und gibt dem Nutzer das Feedback, dass die Aktion der App nicht ausgeführt werden konnte und welche zusätzlichen Berechtigungen benötigt werden. Der Nutzer kann daraufhin entscheiden, ob er der App diese Berechtigungen gewähren will. Wenn nicht, wird der Datenzugriff abgebrochen und das *Service Feature* in der App übersprungen. Falls der Nutzer der App die Berechtigungen gibt, so wird eine neue *Policy Rule* in der *Privacy Policy* erstellt und die Anfrage der App an den *PDC* darf verarbeitet werden.

Die *PMP* ermittelt, welche Instanz des *PDCs* für die anfragende App genutzt werden soll und leitet die App-ID sowie die Query an diese weiter. In der Basisimplementierung des *PDCs* wird die App-ID nicht benötigt. Allerdings

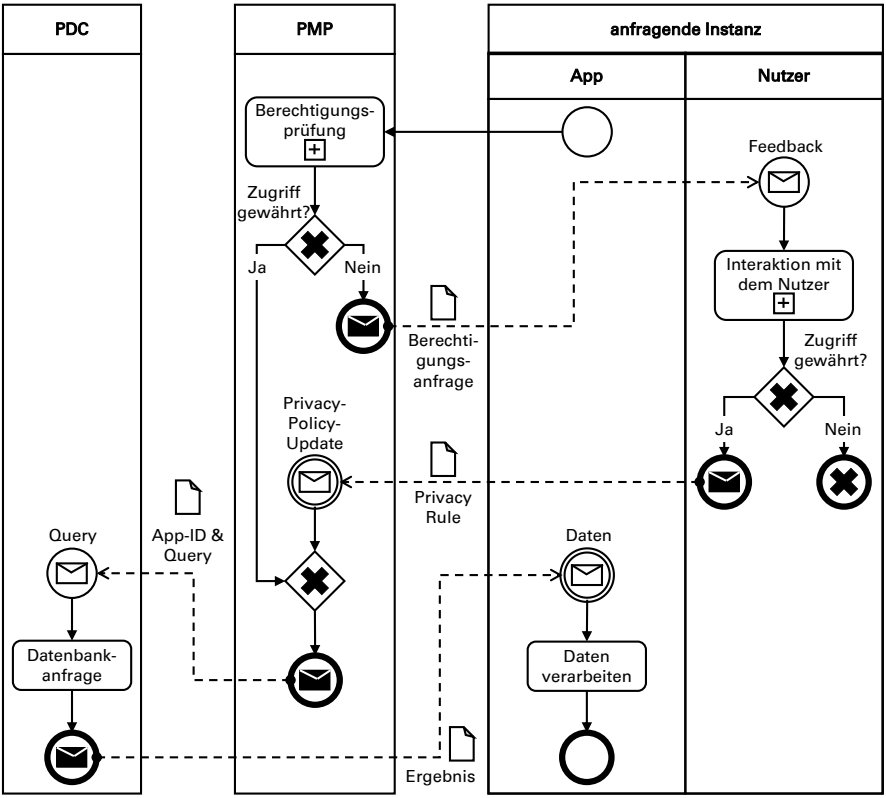


Abbildung 5.2: Prozess des Datenzugriffs über den *PDC*

könnte diese beispielsweise in spezialisierten Versionen des *PDCs* an den in der Query übergebenen Schlüssel angehängt werden, um so jeder App einen eigenen Schlüsselraum zu geben. Der *PDC* führt die Anfrage aus und leitet das Ergebnis an die anfragende App zurück, die dieses entsprechend ihrer Programmlogik weiterverarbeiten kann.

Aus Sicht eines App-Entwicklers liegt der Vorteil des *PDCs* darin, dass dieser weder eine eigene *SQLite*-Datenbank für seine Apps aufsetzen, noch *Content Provider* definieren muss. Aufgrund des generischen Datenmodells, besitzt der *PDC* damit eine hohe Wiederverwendbarkeit. Benötigt ein Entwickler ein anderes Datenmodell, so kann er dieses auf seinen Anwendungsfall anpassen,

um den *PDC* zu spezialisieren – es dürfen beliebig viele Varianten eines *PDCs* parallel betrieben werden. Da der *PDC* von jeder beliebigen App verwendet werden kann, stellt er für App-Entwickler daher eine sehr einfache Möglichkeit dar, um Daten zwischen Apps austauschen zu können. Für den Datenaustausch muss eine App auch nicht berücksichtigen, dass ggf. keine Gegenstelle existiert, z. B. da diese vom Nutzer deinstalliert wurde. Die *PMP* stellt sicher, dass eine App nur dann *Service Features* ausführen kann, die einen *PDC* voraussetzen, wenn eine solche *Resource* installiert ist. Ansonsten werden diese Codeblöcke automatisch übersprungen.

Aus Sicht des Nutzers steht beim *PDC* der verbesserte Datenschutz in Vordergrund. Obwohl der Datenaustausch zwischen Apps erheblich vereinfacht wurde, erfolgt dieser nicht unkontrolliert. Über die *Privacy Settings* hat der Nutzer feingranulare Konfigurationsmöglichkeiten, um den Zugriff auf den *PDC* und damit die Daten einzuschränken. Neben dem Datenbankschema können auch die *Privacy Settings* spezialisiert werden, um dem Nutzer dadurch noch bessere Kontrollmöglichkeiten zu geben. Werden die App-IDs vom *PDC* genutzt, um separate Schlüsselräume zu generieren, so stellt dies ein erster Schritt in Richtung Datenpartitionierung dar, die zur Erfüllung der Schutzziele bezüglich Datensicherheit benötigt wird.

5.2 Eigenschaften des SDCs

Der *SDC* basiert auf dem *PDC*, erweitert diesen aber um zusätzliche Datensicherheitsfeatures. Hierzu führt der *SDC* ein erweitertes, aber dennoch generisches Datenschema ein (Abschnitt 5.2.1). Mit diesem Datenbankschema kann der Nutzer (und auf Wunsch auch der App-Entwickler) feingranular festlegen, welche Daten mit welcher App geteilt werden sollen. Während dies im *PDC* nur auf Containerebene, d. h. für die vollständige Datenbank, möglich ist, können im *SDC* Zugriffsrechte auf Tupelebene vergeben werden (Abschnitt 5.2.3). Um illegale Datenzugriffe vollständig ausschließen zu können, verschlüsselt der *SDC* alle darin gespeicherten Daten (Abschnitt 5.2.2). Sollte das Smart Device oder die mobile Plattform auf irgendeine Weise kompromittiert werden, stellt der *SDC* einen Mechanismus zur Verfügung, mit dem alle im *SDC* gespeicherten Daten zuverlässig und nachhaltig unlesbar und somit für den Angreifer wert-

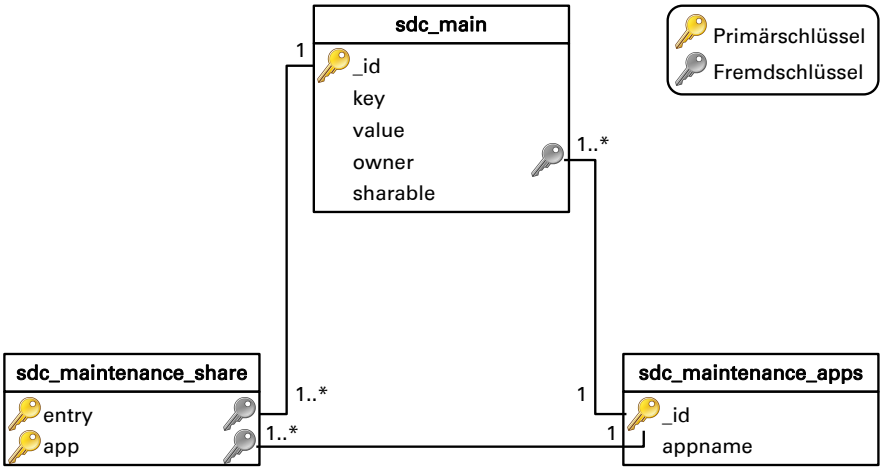


Abbildung 5.3: Datenbankschema des *SDCs* (in Anlehnung an [SM15])

los gemacht werden (Abschnitt 5.2.4). Um neben einer logischen Trennung der Daten, wie es im *PDC* mit den getrennten Schlüsselräumen möglich ist, auch eine physische Trennung der Daten zu ermöglichen, unterstützt der *SDC* eine Partitionierung der Daten (Abschnitt 5.2.5). Diese Partitionierung kann auch dazu genutzt werden, um die Partitionen auf spezielle Anwendungsfälle anzupassen. Hierzu kann beispielsweise die Art der Verschlüsselung oder das Datenschema geändert werden (Abschnitt 5.2.6). Diese sechs Eigenschaften werden im Folgenden näher beleuchtet.

5.2.1 Generisches Datenbankschema

Nach außen hin besitzt der *SDC* in seiner Basisimplementierung genau wie der *PDC* ein generisches Schlüssel-Wert-Datenbankschema. Intern wird dieses Schema um weitere Tabellen und Attribute erweitert, um ebenfalls Informationen zu den gespeicherten Daten hinterlegen zu können. Zu diesen Informationen gehören u. a., woher ein Datum stammt und ob ein Datum mit anderen Apps geteilt werden darf. Dieses interne Datenbankschema des *SDCs* ist in Abbildung 5.3 dargestellt.

Die eigentlichen Nutzdaten werden in der *sdc_main*-Tabelle gespeichert. Diese bestehen aus einem Schlüssel-Wert-Paar, repräsentiert im *key*- und *value*-Attribut. In der Basisimplementierung werden die Daten in ihrer String-Repräsentation gespeichert, um beliebige Datentypen unterstützen zu können. Die Schnittstelle des *SDCs* ist ähnlich reduziert, wie die des *PDCs*. Lediglich eine zweite `put`-Methode wird eingeführt, die eine Liste von Apps als Parameter übergeben bekommt. Grundsätzlich hat im *SDC* nur der Datenerzeuger Zugriff auf seine Daten. Dieser Listenparameter gibt daher an, mit welchen weiteren Apps ein Datum geteilt werden darf. Für den *SDC* gilt, genau wie für den *PDC*, dass das Datenschema sowie die Schnittstelle beliebig erweitert werden kann, wenn dies für einen Anwendungsfall nötig ist.

Die *PMP* liefert beim Zugriff auf den *SDC* die restlichen Metadaten. In dem *owner*-Attribut wird der Datenerzeuger eines Eintrags abgespeichert. Die *PMP* übergibt einer *Resource* bei jedem Aufruf einer ihrer öffentlichen Methode die ID der aufrufenden App. Dieser Identifikator wird bei der Verwendung der `put`-Methode als Kennzeichnung des Datenerzeugers verwendet. Da die App-ID zwar ein eindeutiger, aber kein eidetischer Bezeichner ist, wird zusätzlich der App-Name in der *sdc_maintenance_apps*-Tabelle abgespeichert (*appname*-Attribut). Ein Eintrag in diese Tabelle wird für jede App, die bei der *PMP* registriert ist, erzeugt. Der sprechende Bezeichner wird für das Feedback an sowie die Interaktion mit dem Nutzer benötigt.

Damit ein Tupel von einer anderen App als der, die die Daten erzeugt hat, angefragt werden kann, muss dieses zunächst freigegeben werden. Dies geschieht über das boolesche Attribut *sharable* in der Haupttabelle. Für jedes Tupel, das das *sharable*-Flag besitzt, existieren in der *sdc_maintenance_share*-Tabelle Einträge. In dieser Brückentabelle werden Datentupel mit Apps, die diese verarbeiten dürfen, verknüpft.

5.2.2 Datenverschlüsselung

Die Berücksichtigung, wer Datenerzeuger ist und mit wem dieser seine Daten teilt, ist allerdings wenig hilfreich, wenn Angreifer über Schadsoftware illegal Zugriff auf die Daten erhalten. Shmueli et al. [SVEG09] beschreiben unterschiedliche Angriffsvektoren, die sich gegen Datenbanken richten. Dabei beziehen sie die unterschiedlichsten Angreifer (z. B. *Insider*, übertragen auf den

SDC wären dies Apps, oder *Eindringlinge*, die beim *SDC* durch physische Zugriffe auf das Smart Device repräsentiert werden) sowie diverse Angriffstechniken (z. B. Angriffe auf die laufende Datenbank oder Angriffe auf die Dateien, in denen die Daten der Datenbank gespeichert sind) in ihre Analyse mit ein. Das Ergebnis dieser Untersuchung zeigt, dass ein umfassender Schutz, der sowohl illegale Zugriffe als auch Datenmanipulationen zuverlässig abwehrt, nur dann gewährleistet werden kann, wenn die Datenbank vollständig verschlüsselt wird. Daher verschlüsselt der *SDC* alle darin gespeicherten Daten[5].

Ein so umfassender Schutz hat allerdings seinen Preis. Durch die Ver- und Entschlüsselungsoperationen büßt eine Datenbank an Performanz ein. Insbesondere, wenn bei jedem Zugriff die vollständige Datenbank entschlüsselt werden muss, können diese Einbußen gewaltig sein. Eine Aufteilung der gespeicherten Daten könnte diesem entgegenwirken. Auch die Qualität des Kryptographiealgorithmus hat Einfluss auf die Kosten der Ver- und Entschlüsselung. Da nicht alle Daten gleich schützenswert sind, wäre die Einführung von Schutzzonen sinnvoll, die auf Partitionen der Datenbank abgebildet werden. In jeder Schutzzone käme ein anderer Kryptographiealgorithmus zum Einsatz, um für jedes Datum ein optimales Kosten-Sicherheit-Verhältnis zu erzielen. Der *SDC* berücksichtigt diese Erkenntnisse; Abschnitt 5.2.5 beschreibt, wie die Partitionierung von Daten im *SDC* realisiert werden kann und Abschnitt 5.2.6 befasst sich mit der Einführung von Schutzzonen.

Da das Hauptaugenmerk des *SDCs* auf der Datensicherheit liegt, müssen illegale Datenzugriffe zuverlässig ausgeschlossen werden. Zu diesem Zweck wird jede *SDC*-Instanz vollständig chiffriert. Im Prototyp kommt hierfür der Advanced Encryption Standard (AES)-256-Verschlüsselungsalgorithmus zum Einsatz. Der AES-256-Algorithmus ist von der National Security Agency (NSA) für den Schutz von sensitiven Daten und Daten, die unter Verschluss stehen, offiziell zugelassen [CNSS13]. Da auf Smart Devices in der Regel Daten gespeichert werden, die wesentlich weniger sensibel sind, genügt dieser Algorithmus auch den Anforderungen des *SDCs*. Allerdings kann der AES-256-Algorithmus durch jedes beliebige symmetrische Kryptographieverfahren ersetzt werden. Obwohl dieser Kryptographiealgorithmus sehr rechenintensiv ist, zeigen Benchmark-

[5] Unter Android werden Datenbanken standardmäßig vollständig unverschlüsselt als Klartext gespeichert.

Tests, dass der Rechen-Overhead in einem vertretbaren Rahmen liegt (siehe Abschnitt 7.2).

Der *SDC* ist die einzige Instanz, die einen direkten Zugang auf die Datenbank benötigt. Apps, die Zugang zu den darin gespeicherten Daten benötigen, müssen eine Anfrage über die *PMP* an den *SDC* stellen. Dieser führt daraufhin die Anfrage aus und sendet die Ergebnisse an die App zurück. Daher benötigt ausschließlich der *SDC* den Schlüssel, um die Datenbank dechiffrieren zu können. Ein Austausch des Schlüssels mit einer App ist daher nicht nötig. Asymmetrische Kryptographieverfahren sind daher nicht sinnvoll. Allerdings schließt der *SDC* diese nicht aus; er würde in diesem Fall sowohl den privaten als auch den öffentlichen Schlüssel verwalten.

Der *SDC* muss sich vor der Dechiffrierung nicht darum kümmern, ob eine App überhaupt Zugang zu der Datenbank erhalten darf. Eine solche Überprüfung führt die *PMP* bei jedem Zugriff auf die Schnittstelle einer *Resource* im Vorfeld durch.

5.2.3 Verbesserter Mechanismus zum Datenaustausch zwischen Apps

Der *PDC* führt bereits einen Mechanismus für Android ein, der einen einfachen Austausch von Daten zwischen Apps ermöglicht. Das Problem bei diesem Ansatz liegt in der grobgranularen Zugriffsbeschränkung. Der *SDC* erweitert daher die Zugriffskontrolle. Abbildung 5.4 stellt in einer an Unified Modeling Language (UML) angelehnten Notation dar, wie dieser Kontrollmechanismus eine Datenanfrage von einer App verarbeitet. Hierbei handelt es sich ausschließlich um die Überprüfungen, die der *SDC* selbst ausführt. Der vollständige Prozess eines Datenzugriffs über den *SDC* ist in Abschnitt 5.3 beschrieben.

Nachdem die *PMP* einer App den Zugriff auf den *SDC* gestattet hat, dechiffriert der *SDC* mit seinem Schlüssel die benötigte Datenbank. Anschließend prüft der *SDC*, ob die anfragende App bereits bei ihm registriert ist. Prinzipiell registriert die *PMP* alle Apps beim *SDC*, wenn diese installiert werden. Da *Resource Groups*, und damit auch der *SDC*, allerdings nachträglich der *PMP* hinzugefügt werden können, kann es Apps geben, die nicht automatisch beim *SDC* registriert wurden. Dies betrifft alle Apps, die vor dem *SDC* installiert wurden. Daher führt der *SDC* diese zusätzliche Überprüfung durch und legt für unregis-

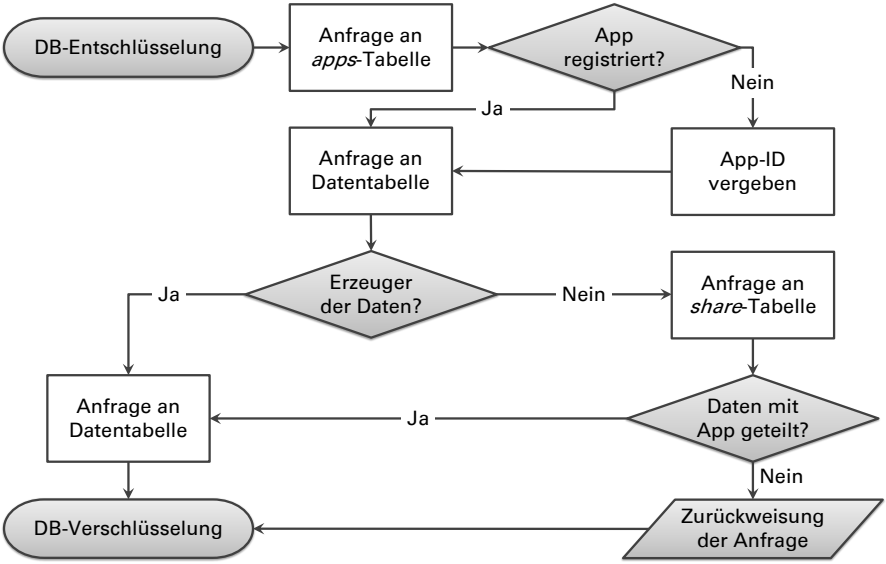


Abbildung 5.4: *SDC*-Anfragealgorithmus (in Anlehnung an [SM15])

trierte Apps die entsprechenden Einträge in der *sdc_maintenance_apps*-Tabelle an.

Mittels der systemweit eindeutigen App-ID wird überprüft, ob die anfragende App der Datenerzeuger ist. Standardmäßig ist nur dieser dazu berechtigt, auf seine Daten zuzugreifen. Für den Datenerzeuger werden die Operationen und Anfragen auf der Datenbank ausgeführt und die Datenbank wird wieder verschlüsselt.

Der unkontrollierte Informationsfluss zwischen Apps, wie es bei den *Content Providern* unter Android der Fall ist, stellt eine Bedrohung für die Datensicherheit dar [OMEM12]. Auch die Zugriffskontrolle über die *PMP* ist zu grobgranular, da diese auf die Verwendungsart einer *Resource* abzielt und nicht direkt auf deren Daten. Daher führt der *SDC* für den Datenaustausch, d. h. wenn es sich bei der anfragenden App nicht um den Datenerzeuger handelt, einen sichereren Modus Operandi ein. Kann der angefragte Datensatz mit anderen Apps geteilt werden (*sharable*-Flag), prüft der *SDC* über die *sdc_maintenance_share*-Tabelle, ob dies auch für die anfragende App der Fall ist. Da über diese Tabelle für jedes Schlüssel-Wert-Paar individuell eine Menge an Apps definiert

werden kann, mit denen das Tupel geteilt werden kann, können Zugriffsrechte vom *SDC* feingranular vergeben werden. Alle geteilten Daten werden in einer Ergebnismenge zusammengefasst und an die App zurückgegeben, nachdem die Datenbank wieder verschlüsselt wurde. Nur Datenerzeuger können das *sharable*-Flag sowie die Einträge in der *share*-Tabelle ändern.

Diese Vorgehensweise erfüllt nicht nur die in Abschnitt 1.3 aufgestellten Schutzziele (siehe Abschnitt 7.1), sondern ist auch sehr performant (siehe Abschnitt 7.2).

5.2.4 Zuverlässiges Löschen von Daten

Werden Daten von einem Datenspeicher gelöscht, so können diese in der Regel sehr leicht wiederhergestellt werden. Eine Untersuchung zu diesem Thema von Joukov, Papaxenopoulos und Zadok [JPZ06] zeigt, dass selbst viele Spezialsysteme zur sicheren Datenlöschung unzuverlässig arbeiten. Sichere Verfahren verschlüsseln zunächst die Daten und vernichten anschließend den Schlüssel oder sie überschreiben den Speicherbereich, auf dem die Daten zuvor gespeichert wurden, mehrfach [WKS08].

Im Bereich von Datenbanken zeigt sich, dass das sichere, d. h. unwiederbringliche Löschen von Daten besonders schwierig ist. Im Cache des Datenbanksystems finden sich häufig noch viele Daten und forensische Analysen sind in der Lage hieraus selbst verschlüsselte Daten auszulesen [GAS10]. Jeon et al. [JBBL12] zeigen, wie einfach sich gelöschte Daten einer *SQLite*-Datenbank wiederherstellen lassen.

Da der *SDC* zur Speicherung von sensiblen Daten genutzt wird, muss dieser für den Fall einer Kompromittierung einen Mechanismus zur sicheren Löschung der Daten anbieten. Dieser Mechanismus wird als *Kill Switch* bezeichnet. Da einzelne Daten nicht zuverlässig in einem Datenbanksystem gelöscht werden können [SML07] und der *Kill Switch* nur für den Ernstfall gedacht ist, macht dieser daher die vollständige Datenbank unbrauchbar. Da eine *SQLite*-Datenbank aus einer einzigen Datei besteht, muss diese sicher gelöscht werden.

Ein mehrfaches Überschreiben der *SQLite*-Datei kommt auf Smart Devices nicht in Frage, da diese Geräte zur nichtflüchtigen Speicherung NAND-Flash-Speicher verwenden. Diese Technologie speichert Daten nicht sequentiell ab. Ein mehrfaches Überschreiben würde daher voraussetzen, dass der *Kill Switch*

alle Speicheradressen kennt, an denen die *SQLite*-Datei gespeichert ist, und diese direkt ansteuern kann. Eine solche Verwendung ist für NAND-Flash-Speicher aber aus Performanzgründen nicht vorgesehen.

Daher kommt im *SDC* ein kryptographisches Löschverfahren zum Einsatz. Die Datenbank des *SDCs* ist ohnehin vollständig verschlüsselt. Im Falle einer Kompromittierung muss daher nur der Schlüssel sicher gelöscht werden, um die Daten des *SDCs* unlesbar zu machen. Da der *SDC* den Schlüssel zu keinem Zeitpunkt an andere Instanzen weitergibt, können von dem Schlüssel auch keine Kopien existieren. Zum sicheren Löschen des Schlüssels kann auf ein Verfahren, wie es von Lee et al. [LHC+08] vorgestellt wird, zurückgegriffen werden. Mit dieser Technik lassen sich kleine Dateien auch von NAND-Flash-Speicher zuverlässig löschen; für die Löschung einer ganzen Datenbanken ist sie nicht geeignet.

Durch die Anwendung des *Kill Switch* werden allerdings alle Daten des *SDCs* gelöscht. Manchmal sollen aber nur Teile der Datenbank gelöscht werden, z. B. da sowohl hochsensible Daten als auch unbedenkliche Daten darin gespeichert werden und Erstere bereits beim leisesten Verdacht, dass das System kompromittiert worden sein könnte, gelöscht werden müssen. In diesem Fall kann ausgenutzt werden, dass der *SDC* eine Partitionierung der Daten unterstützt (siehe Abschnitt 5.2.5). Jede Partition besitzt einen eigenen Schlüssel, wodurch sich einzelne Partitionen mit dem *Kill Switch* löschen lassen, ohne die anderen zu beeinflussen.

Waizenegger [Wai15] schlägt eine hierarchische Ordnung der Schlüssel in einer Baumstruktur vor. Die Schlüssel der Partitionen liegen auf der untersten Ebene im Baum. Auf den höheren Ebenen im Baum befinden sich zusätzliche, neu eingeführte Schlüssel, die dafür benötigt werden, um darunterliegende Schlüssel verwenden zu können. Mit diesem System können einzelne Partitionen oder ganze Cluster von Partitionen über die Löschung eines einzelnen Schlüssels unlesbar gemacht werden [Wai17]. Daten können über eine solche hierarchische Schlüsselstruktur effizient und zuverlässig gelöscht werden [RRBC13]. In der Basisimplementierung des *SDCs* ist ein solches Schlüsselmanagement nicht vorgesehen. Wird allerdings im *SDC* die Datenpartitionierung genutzt, so kann auf diese Technik zur feingranularen Löschung zurückgegriffen werden.

Über den *Kill Switch* können die Daten des *SDCs* auch mittels Fernlöschung unlesbar gemacht werden. Hierfür benötigt der Nutzer lediglich einen Android-Service, der auf ein Signal hin (beispielsweise per Short Message Service (SMS)) den *Kill Switch* auslöst. Dadurch benötigt er nicht einmal physischen Zugriff auf sein Smart Device und er kann selbst, wenn das Gerät gestohlen wurde, sicherstellen, dass seine vertraulichen Daten nicht von Dritten genutzt werden können.

5.2.5 Datenpartitionierung

In der Basiskonfiguration des *SDCs* liegen alle darin gespeicherten Daten in einer einzigen großen *sdc_main*-Tabelle. Dies kann aus zwei Gründen problematisch sein. Einerseits muss das Datenschema dieser Tabelle so generisch gehalten sein, dass jede beliebige App damit umgehen kann, und andererseits muss der Nutzer dadurch vielen Apps die Berechtigung erteilen, auf den *SDC* zuzugreifen. Auch wenn der *SDC* selbst nochmals feingranulare Zugriffskontrollen durchführt, entspricht eine solche Datenspeicherung nicht dem *Least-Privilege*-Prinzip.

Daher kann die Erweiterbarkeit der *PMP* dahingehend ausgenutzt werden, dass mehrere Instanzen des *SDCs* (implementiert als individuelle *Resources*) zum Einsatz kommen, z. B. als Varianten mit unterschiedlichen Datenschemata oder unterschiedlichen Kryptographiealgorithmen (siehe Abschnitt 5.2.6). In jeder *SDC-Resource* werden nur die Daten von zuvor ausgewählten Apps gehalten. Dadurch beschränkt sich die Zugriffsberechtigung dieser Apps bereits von vornherein auf die Daten, die in ihrer *SDC*-Instanz gespeichert sind. Da für jede *SDC-Resource* eine eigene *SQLite*-Datenbank und damit eine separate *SQLite*-Datei verwendet wird, sinken mit der Aufteilung der Daten automatisch auch die Kosten für die Verschlüsselung und der *Kill Switch* kann feingranularer arbeiten.

Tabelle 5.1 zeigt, welche Partitionierungsmodelle vom *SDC* unterstützt werden und welche Auswirkungen dies auf die Sicherheit der Daten sowie die Möglichkeit Daten zwischen Apps auszutauschen hat.

$[1 : n]$. Im einfachsten Modell wird eine einzige Instanz des *SDCs* von beliebig vielen Apps verwendet. Alle Apps speichern ihre Daten in einer gemeinsamen Datenbasis. Dadurch profitieren alle Apps von den verbesserten Datenaustausch-

# SDCs	# Apps	Eigenschaften	
		Sicherheit	Datenaustausch
1	1	App hat eigenen *SDC*	kein Datenaustausch möglich
1	n	alle Apps arbeiten auf gemeinsamen *SDC*	vollständiger Datenaustausch möglich
m	1	App verteilt Daten auf mehrere private *SDCs*	*SDCs* stehen App exklusiv zur Verfügung
m	n	mehrere *SDCs* (z. B. mit unterschiedlichen Datenschemata oder Sicherheitsstufen) stehen mehreren Apps zur Verfügung	Datenaustausch nur mit Apps innerhalb gemeinsamer *SDCs* möglich

Tabelle 5.1: Partitionierungsmodelle des *SDCs*

Features des *SDCs*. Allerdings muss der Nutzer hierzu über die *PMP* einer App zunächst vollen Zugriff auf den *SDC* gewähren. Dieser Zugriff kann zwar über den *SDC* feinjustiert werden, dennoch stellt dies eine potentielle Gefahrenquelle dar. Außerdem muss das Datenschema möglichst generisch gehalten werden (z. B. ein Key-Value-Schema), um alle Apps unterstützen zu können. Auch steigt die Größe des Datenspeichers des *SDCs* rasch an, was sich negativ auf dessen Performanz auswirkt.

[1 : 1]. Eine andere Verwendungsart für den *SDC* besteht darin, dass nur eine einzige App diesen benutzen darf, bzw. dass für jede App eine eigene private Instanz des *SDCs* bereit steht. In diesem Modell kann die App beispielsweise das Datenschema oder den Kryptographiealgorithmus auf ihre Bedürfnisse anpassen, wodurch eine Performanzsteigerung bei den Anfragen erzielt werden kann. Da nur eine App ihre Daten in der jeweiligen *SDC*-Instanz ablegt, wächst deren Datenspeicher auch nicht so schnell an, was sich ebenfalls positiv auf die Performanz auswirkt. Anfragen anderer Apps werden bereits von der *PMP* blockiert, bzw. diese werden automatisch mit der für sie vorgesehene Instanz

des *SDCs* verbunden, wodurch sich die Datensicherheit erhöht. Ein Austausch der Daten mit anderen Apps ist allerdings in diesem Modell nicht möglich.

[$m : 1$]. Muss eine App mit vielen heterogenen Daten umgehen, so bietet sich ein Modell an, in dem einer App viele private *SDC*-Instanzen zur Verfügung stehen. Die App kann selbstständig die Partitionierung ihrer Daten vornehmen und dadurch beispielsweise festlegen, welche Schutzzone für ein Datum geeignet ist. Da die *SDC*-Instanzen der App exklusiv zur Verfügung stehen, ist dies hinsichtlich der Datensicherheit positiv, aber ein Austausch der Daten mit anderen Apps ist nicht möglich.

[$m : n$]. Im letzten Partitionierungsmodell stehen Apps mehrere *SDC*-Instanzen zur Verfügung, jedoch nutzen sie diese nicht exklusiv. Dadurch können Apps, die auf der gleichen Instanz ihre Daten halten, diese austauschen. Gleichzeitig hat auf jede Instanz nur eine beschränkte Menge an Apps Zugriff. Da jede Instanz auf einen speziellen Anwendungsfall zugeschnitten werden kann (siehe Abschnitt 5.2.6), bietet dieser Ansatz eine hohe Flexibilität bezüglich Datensicherheit und Datenaustauschmöglichkeiten.

5.2.6 Anpassbarkeit auf unterschiedliche Anwendungsfälle

Da die Verwendungsmöglichkeiten für die *PMP* sowie den *SDC* sehr vielseitig sind (siehe Kapitel 6), müssen diese beiden eine hohe Flexibilität besitzen. Die *PMP* erreicht dies u. a. durch ihre Erweiterbarkeit (siehe Abschnitt 4.2.5). Das Anpassungspotential des *SDCs* liegt in dessen Datenbankschema und dem verwendeten Kryptographiealgorithmus. Durch diese beiden Eigenschaften können mehrere Implementierungsvarianten des *SDCs* erstellt werden, von denen jede auf einen speziellen Anwendungsfall angepasst ist. Die App-Entwickler können dadurch beispielsweise wählen, welches Partitionierungsmodell, welcher Kryptographiealgorithmus oder welches Datenbankschema für ihre App am geeignetsten ist. Jede Variante des *SDCs* stellt implementierungstechnisch eine neue *Resource* dar, für die sogar Schnittstellenerweiterungen sowie neue *Privacy Settings* möglich sind.

```
/* ursprüngliche Query */
SELECT value
FROM sdc_main
WHERE key = '123';

/* umgeformte Query */
SELECT m.value
FROM sdc_main m,
     sdc_maintenance_apps a,
     sdc_maintenance_share s
WHERE m.key = '123' AND (
     (m.owner = a._id AND a.appname = ?APP?) OR
     (m.sharable = 1 AND m._id = s.entry AND
      s.app = a._id AND a.appname = ?APP?));
```

Quelltext 5.2: Query-Umformung des *SDCs*

5.3 Funktionsweise des SDCs als PMP-Resource

Da der *SDC* grundsätzlich auf dem *PDC* basiert, läuft der Datenzugriff unter Verwendung des *PDCs* und des *SDCs* sehr ähnlich ab. Allerdings sind ein paar wesentliche Anpassungen an dem Datenzugriffsprozess nötig, um die in Abschnitt 5.2 beschriebenen Datensicherheitsfeatures realisieren zu können.

Die initialen Schritte unterscheiden sich dabei nicht von denen des *PDC*-Datenzugriffsprozess. Die anfragende App kontaktiert die *PMP*, damit diese den Zugriff auf den *SDC* genehmigt. Besitzt die App die benötigte Berechtigung nicht, so wird der Nutzer von der *PMP* kontaktiert und kann bei Bedarf die *Privacy Policy* anpassen. Wurde der Zugriff schließlich genehmigt, unterscheiden sich die beiden Datenzugriffsprozesse.

Falls ein Partitionierungsmodell mit mehr als einer *SDC*-Instanz zum Einsatz kommt, muss die *PMP* zunächst bestimmen, an welche Instanz die Anfrage gerichtet wird. Da jede Instanz auf eine *Resource* abgebildet wird, kann die *PMP* die hierfür benötigten Informationen direkt aus der Berechtigungsanfrage auslesen. Anschließend bindet sie die richtige Instanz an die anfragende App und leitet die Query sowie die App-ID weiter.

Zunächst wird allerdings die Query umgeformt. Beispielsweise muss sichergestellt werden, dass nur Tupel berücksichtigt werden, die von der anfragenden App erzeugt oder mit dieser geteilt wurden. In Quelltext 5.2 ist für eine einfache Anfrage dargestellt, wie diese vom *SDC* umgeformt wird. Die Zugriffsberechtigung wird in den Zeilen 12 bis 14 realisiert. Der Parameter `?APP?` wird von der *PMP* über die App-ID ersetzt.

Da die Daten im *SDC* ausschließlich verschlüsselt vorliegen, muss dieser die interne *SQLite*-Datenbank zunächst entschlüsseln, bevor eine Anfrage verarbeitet werden kann. Hierfür benötigt der *SDC* keine weiteren Daten. Der benötigte Schlüssel liegt im *SDC* vor und für jede App, deren Anfrage an den *SDC* weitergeleitet wird, hat die *PMP* sichergestellt, dass die Anfrage nicht gegen die *Privacy Policy* verstößt.

Anschließend kann die Anfrage durchgeführt werden. Dadurch, dass die Query bereits umgeformt wurde, berücksichtigt der *SDC* automatisch die Zugriffsrechte auf Tupelebene. Das Anfrageergebnis beinhaltet ausschließlich die Daten, die entweder von der anfragenden App selbst erzeugt oder explizit mit ihr geteilt wurden.

Allerdings können über *Privacy Settings* weitere Manipulationen an den Daten nötig sein. In der Basisimplementierung des *SDCs* sind hierfür keine speziellen *Privacy Settings* vorgesehen. Im Rahmen einer Spezialisierung des *SDCs* auf einen konkreten Anwendungsfall, können auf diese Weise Filterungen oder Aggregatfunktionen beschrieben werden. Diese Datenverschleierungsoperationen sind im *SDC* definiert. Aus diesem Grund kann die *PMP* diese bei der Umformung der Query nicht direkt mitberücksichtigen.

Da der Fokus des *SDCs* primär auf der Datensicherheit liegt, wird die Datenbank nur so lange wie unbedingt nötig entschlüsselt. Daher findet unmittelbar, nachdem die Anfrage verarbeitet wurde, eine erneute Verschlüsselung der *SQLite*-Datenbank statt. Erst wenn die *SQLite*-Datenbank wieder vollständig verschlüsselt wurde, werden die bereinigten Ergebnisse an die anfragende App übermittelt, die diese weiterverarbeiten kann. Der überarbeitete Datenzugriffprozess des *SDCs* ist in Abbildung 5.5 in einer an BPMN angelehnten Notation dargestellt.

Im Vergleich zu Abbildung 5.2 erkennt man beim überarbeiteten Datenzugriffprozess des *SDCs*, dass sich für die *PMP* und die anfragende Instanz

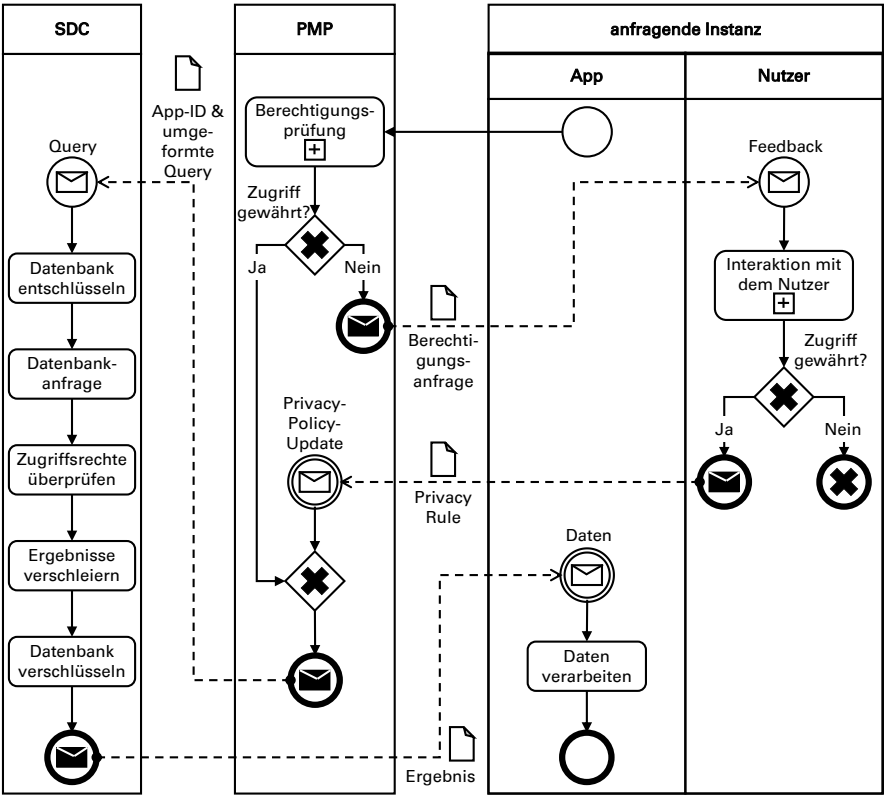


Abbildung 5.5: Prozess des Datenzugriffs über den *SDC* (in Anlehnung an [SM16a])

keine Änderungen ergeben. Lediglich der Anfrageprozess innerhalb des *SDCs* unterscheidet sich von dem das *PDCs*.

5.4 Einführung von Nutzerkonten und Datenaustausch zwischen Nutzern im SDC+

Die Kombination aus *PMP* und *SDC* stellt bereits einen großen Schritt in Richtung holistisches Sicherheitssystem dar, wie die in Kapitel 7 beschriebenen Untersuchungsergebnisse belegen. Der *SDC* hat allerdings zwei Schwachstellen. Die zu schützenden Daten liegen im *SDC* nur verschlüsselt vor. Ein Angreifer kann diese daher nur dann nutzen, wenn er auch den Schlüssel besitzt. Da allerdings sowohl die Daten als auch der Schlüssel im *SDC* gespeichert werden, kann der Schutz daher nur dann aufrecht erhalten werden, wenn nicht der *SDC* als ganzes kompromittiert wird.

Das andere Problem besteht darin, dass die Zugriffsrechte auf die im *SDC* gespeicherten Daten an Apps gebunden sind. Wird eine App allerdings von mehreren Nutzern verwendet, so hat jeder Nutzer automatisch Zugriff auf die Daten aller anderer Nutzer der gleichen App. Außerdem soll der *SDC* in erster Linie gegen Datenmissbrauch durch Schadsoftware, die auf dem Smart Device des Nutzers installiert ist, schützen. Ein Angreifer könnte daher die Daten aus dem *SDC* auslesen, sobald er physischen Zugriff auf das Smart Device des Nutzers erhält.

Der *SDC+* erweitert den *SDC* daher dahingehend, dass er Nutzerkonten zusammen mit einem sicheren Mechanismus zur Eingabe von Anmeldeinformationen einführt sowie mehrere *SQLite*-Datenbanken innerhalb einer *SDC+*-Instanz verwaltet. Durch die Nutzerkonten ist es möglich, die Zugriffsrechte an eine Kombination aus Nutzer und App zu binden, wodurch jeder Nutzer nur Zugriff auf seine eigenen Daten hat – ohne die korrekten Anmeldeinformationen erhält ein möglicher Angreifer keinen Zugriff. Auch können Nutzer auf diese Weise Daten nicht nur mit anderen Apps, sondern auch Daten gezielt mit anderen Nutzern tauschen. Durch die Partitionierung, d. h. die Aufteilung der Daten auf mehrere *SQLite*-Datenbanken, innerhalb einer *SDC+*-Instanz besitzt der *SDC+* die gleichen Performanzvorteile wie die Nutzung mehrerer *SDC*-Instanzen (siehe Abschnitt 5.2.5). Allerdings kann im *SDC+* ein Datenaustausch auch über Partitionsgrenzen hinweg stattfinden, da diese innerhalb der gleichen Instanz laufen. Jede Partition ist mit einer anderen Schlüsselkombination verschlüsselt. Die verwendeten Schlüssel werden auf den Nutzer, die *PMP* und den *SDC+*

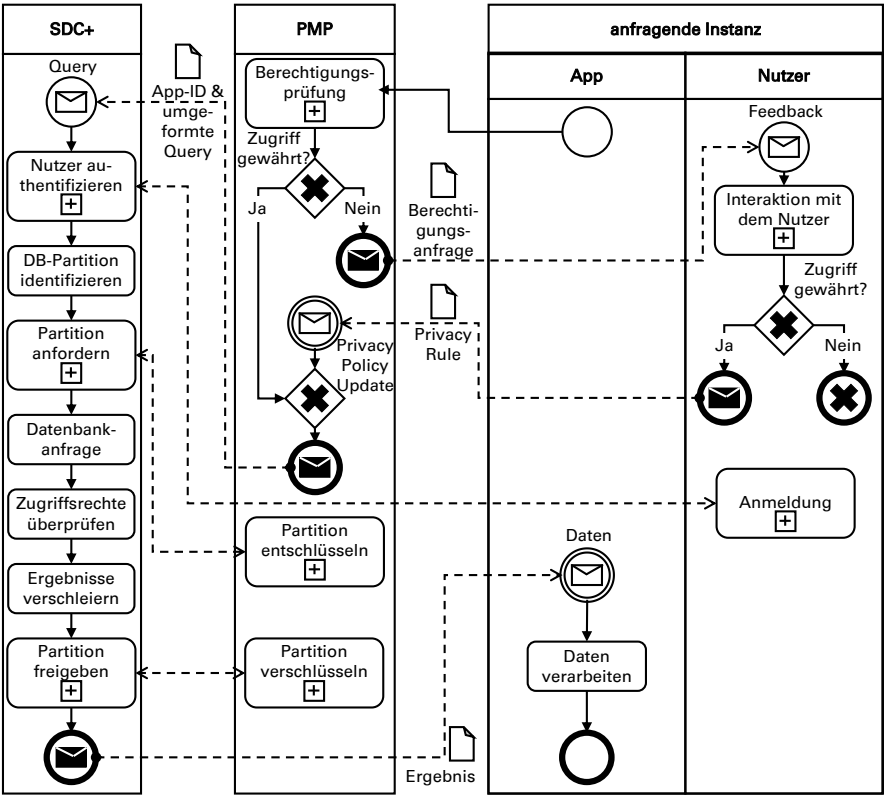


Abbildung 5.6: Prozess des Datenzugriffs über den *SDC+*

aufgeteilt. Dadurch sind die Daten im *SDC+* selbst dann noch geschützt, wenn eine dieser drei Parteien kompromittiert wird. Die Funktionsweise des *SDC+* wird im Folgenden detailliert.

In Abbildung 5.6 ist der Prozess des Datenzugriffs unter Verwendung des *SDC+* in einer an BPMN angelehnten Notation dargestellt. Dieser Prozess ist bis zu dem Punkt, an dem die *PMP* die Anfrage an den *SDC+* weiterleitet, identisch zu dem Datenzugriffsprozess des *SDCs* (siehe Abbildung 5.5). Anschließend authentifiziert der *SDC+* den Nutzer. Hierfür besitzt der *SDC+* eine spezielle Login-Maske. Diese wird im Prozess des *SDC+* gestartet, d. h. die vom Nutzer eingegebenen Anmeldeinformationen werden nicht mit der App, die

auf den *SDC+* zugreift, sondern ausschließlich mit dem *SDC+* geteilt. In einer privaten und von außen nicht zugreifbaren Datenbank des *SDC+* ist hinterlegt, welcher Nutzer Zugriff zu welchen Partitionen, d. h. *SQLite*-Datenbanken, hat. Anschließend fordert der *SDC+* die relevanten Partitionen von der *PMP* an. Dieser zusätzliche Schritt ist nötig, da die *PMP* einen Teil des für die Entschlüsselung der Datenbank nötigen Schlüssels besitzt. Der vollständige Entschlüsselungsprozess ist in Abbildung 5.7 dargestellt und wird im Anschluss detailliert. Nachdem die Datenbanken dem *SDC+* zur Verfügung stehen, führt dieser die Anfrage durch und filtert, genau wie der *SDC*, die Ergebnismenge, wenn dies erforderlich ist. Anschließend werden die Partitionen von der *PMP* und dem *SDC+* wieder verschlüsselt und die Ergebnisse werden an die anfragende App geschickt.

Jede Datenbankpartition ist mit zwei Schlüsseln chiffriert. Einen Generalschlüssel besitzt die *PMP*. Dieser ist für alle Partitionen gültig. Der zweite Schlüssel, der für jede Partition individuell ist, ist nur dem *SDC+* bekannt. Für den Zugriff auf den zweiten Schlüssel sind allerdings die Login-Daten des Nutzers nötig.

Wie in Abbildung 5.7 zu sehen ist, erfolgt die Dechiffrierung der Daten daher in mehreren Schritten. Zunächst muss ein Nutzer seine Anmeldeinformationen eingeben. Der *SDC +* findet damit über die Partitionstabelle die für diesen Nutzer relevanten Datenbanken. Zusätzlich liegen in der Partitionstabelle auch die individuellen Schlüssel. Der *SDC+* leitet anschließend die verschlüsselte Partition an die *PMP* weiter, die die erste Dechiffrierung mit dem Generalschlüssel durchführt. Danach kann der *SDC+* diese mit dem individuellen Schlüssel vollständig entschlüsseln. Die spätere Verschlüsselung erfolgt analog dazu in umgekehrter Reihenfolge.

Durch dieses Vorgehen wird das System gegen drei Angriffsvektoren gehärtet: Gelingt es einem Angreifer an die Login-Daten eines Nutzers zu gelangen (Angriffsvektor (1)), so kann er sich damit auf dem Smart Device des Nutzers authentifizieren und auf alle Daten dieses Nutzers sowie alle Daten, die mit diesem Nutzer geteilt werden, zugreifen. Die Daten der anderen Nutzer sind allerdings weiterhin geschützt. Gelingt es einem Angreifer die *PMP* zu kompromittieren (Angriffsvektor (2)), so hat er Zugriff auf den Generalschlüssel. Damit kann er zwar auf allen Partitionen die erste Dechiffrierung durchführen,

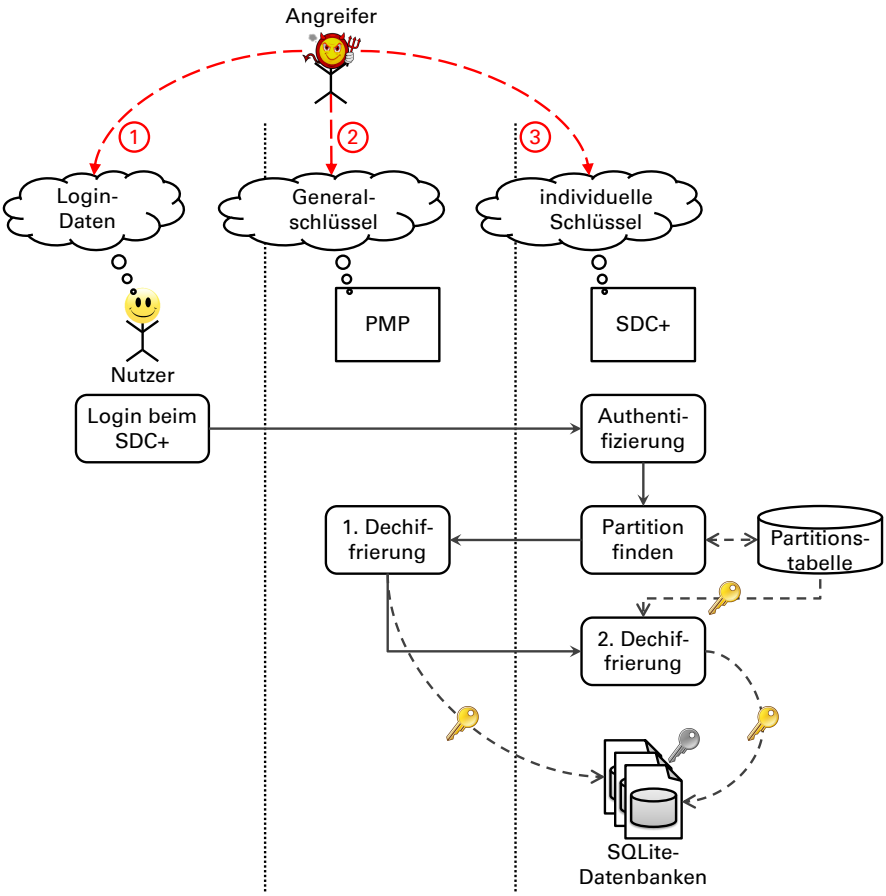


Abbildung 5.7: Dechiffrierungsprozess des *SDC+*

die darin gespeicherten Daten sind allerdings weiterhin verschlüsselt. Gelingt es einem Angreifer den *SDC+* zu kompromittieren (Angriffsvektor (3)), so hat er Zugriff auf die individuellen Schlüssel. Ohne den ersten Dechiffrierungsschritt der *PMP* ist dieser Schlüssel allerdings nutzlos. Daher wird die Datensicherheit mit dem *SDC+* auch dann weitestgehend aufrechterhalten, wenn eine der drei Parteien kompromittiert wird.

5.5 Verwandte Arbeiten

Eine sorgfältige Literaturrecherche ergab, dass es zum Erstellungszeitpunkt der vorliegenden Arbeit keine vergleichbaren Ansätze gibt, die sich mit allen Aspekten des *SDCs* respektive des *SDC+* befassen. Die in diesem Abschnitt diskutierten verwandten Arbeiten sind daher Datensicherheitssysteme (gemäß der in Abschnitt 1.2 gegebenen Definition), die jeweils nur einzelne Teilaspekte aufgreifen. Diese Arbeiten lassen sich in sechs Kategorien einteilen: Kryptographiesysteme, Datenaustauschsysteme, Datenvernichter, Datenpartitionierungssysteme, Nutzerverwaltungssysteme und App-Scanner. Für die Abgrenzung des *SDCs* respektive des *SDC+* zu den verwandten Arbeiten wird berücksichtigt, inwiefern diese die relevantesten Eigenschaften des *SDCs* bzw. des *SDC+* realisieren, die für ein Datensicherheitssystem relevant sind. Da sowohl der *SDC* als auch der *SDC+* auf der *PMP* basieren, zählt hierzu zunächst einmal die Möglichkeit Apps bestimmte **Berechtigungen** zu gewähren respektive zu entziehen. Neu im *SDC* bzw. im *SDC+* hinzugekommen ist, dass die **Verschlüsselung** von Daten möglich ist, ein sicherer **Datenaustausch** zwischen unterschiedlichen Instanzen[6] stattfinden kann, sensible Daten im Fall einer Kompromittierung mittels **Löschfunktion** vernichtet werden können, eine **Partitionierung** von beispielsweise sensiblen und unkritischen Daten möglich ist, die Systeme unterschiedliche **Sicherheitsstufen** unterstützen und dass ein Mehrbenutzerbetrieb über **Nutzerkonten** ermöglicht wird.

Eine **App-Prüfung**, wie sie beispielsweise von Virenscannern durchgeführt wird, wird weder vom *SDC* noch vom *SDC+* unterstützt, obwohl dies ebenfalls ein sinnvolles Feature von Sicherheitssystemen ist. Allerdings zeigen Untersuchungen von Cai und Yap [CY16], dass auf mobilen Plattformen eine solche Überprüfung zur Laufzeit nicht zuverlässig erfolgen kann und diese daher nahezu ineffektiv ist. Einzig eine sorgfältige Überprüfung einer App außerhalb des Smart Devices, z. B. in einem *App-Store*, führt zu überzeugenden Ergebnissen [KGT+15]. Die App-Prüfung müsste demnach bereits erfolgen, bevor die App auf dem Smart Device aufgespielt wird [LMS+14]. Demnach ist die Art der App-Prüfung, die von einem System wie dem *SDC* respektive dem *SDC+*

[6] Eine Instanz kann entweder eine App (*SDC*) oder eine Kombination aus Nutzer und App (*SDC+*) sein.

durchgeführt werden könnte ohnehin nicht zielführend. Der Vollständigkeit halber wird dieses Sicherheitsfeature im Folgenden dennoch mit aufgeführt.

Im Bereich der Sicherheitssysteme gibt es zwar keine Arbeiten, die sich mit sämtlichen oben aufgeführten Aspekten beschäftigen, die Anzahl der Ansätze, die sich mit Teillösungen befassen ist jedoch sehr groß. Daher kann im Folgenden nur ein repräsentativer Auszug aus diesen Arbeiten besprochen werden. Die verwandten Arbeiten werden jeweils einer der obigen Kategorien zugeordnet; befasst sich eine Arbeit mit mehreren Teilaspekten, so wird sie der Kategorie zugeordnet, in der sie die wesentlichsten Beiträge erbringt. Innerhalb einer Kategorie erfolgt die Nennung der Arbeiten in alphabetischer Reihenfolge; aus dieser lässt sich daher keine Wertung ablesen.

5.5.1 Kryptographiesysteme

Attribute-Based Encryption (ABE) [SW05] ermöglicht eine feingranulare Zugriffskontrolle auf persistierte Daten. Zu diesem Zweck wird ein asymmetrisches Kryptographieverfahren dahingehend erweitert, dass sowohl die Ver- als auch die Entschlüsselung mit bestimmten Attributen verknüpft werden können. Abhängig von den Ausprägungen dieser Attribute wird anschließend die kryptographische Operation ausgeführt – selbst wenn zwei Nutzern der korrekte Schlüssel bekannt ist, können diesen mittels ABE unterschiedliche Zugriffsrechte vergeben werden. Beispielsweise können Nutzer mit einem Attribut „Rolle“ verknüpft werden, das festlegt, welche Daten dieser Nutzer verwenden darf. Von ABE gibt es zwei Ausprägungen: Key-Policy ABE (KP-ABE) [GPSW06], bei der die Zugriffsrechte an die privaten Schlüssel der Nutzer gebunden sind und Ciphertext-Policy ABE (CP-ABE) [BSW07], bei der die Zugriffsrechte an die verschlüsselten Daten gebunden sind. Beide Ausprägungen benötigen allerdings sehr viel Rechenleistung, wodurch sie für einen Einsatz in einer mobilen Plattform ungeeignet erscheinen [WZSI14]. Ambrosin, Conti und Dargahi [ACD15] stellen allerdings mit *AndrABEn* eine Implementierung von ABE als Android-Bibliothek vor. In der Theorie stellt ABE einen mächtigen Mechanismus dar, um feingranular Berechtigungen an Instanzen zu vergeben. Um die Bibliothek allerdings nutzen zu können, d. h. damit sie als Berechtigungssystem, zum Datenaustausch oder zur Nutzerverwaltung eingesetzt werden kann, fehlt ein Überbau – dies müsste in jeder App aufs Neue erfolgen.

Viele mobile Plattformen nutzen, genau wie Desktop PCs, Full-disk Encryption (FDE), um die gespeicherten Daten vor illegalen Zugriffen zu schützen. Allerdings schützt diese Art der Verschlüsselung nur, so lange das System ausgeschaltet ist oder sich im Ruhemodus befindet. Wird das System gestartet, so werden die Daten vollständig entschlüsselt und der Schlüssel wird im Speicher gehalten, bis das System wieder heruntergefahren wird. Mobile Plattformen operieren allerdings nach dem *always-on*-Paradigma, d. h. sie werden in der Regel nur gesperrt und nicht komplett ausgeschaltet. Während sie gesperrt sind, laufen im Hintergrund viele Dienste weiter, und die Daten können daher nicht mit FDE verschlüsselt werden. Mit *Deadbolt* stellen Skillen, Barrera und van Oorschot [SBvO13] einen Ansatz vor, mit dem der Zugriff auf den internen Speicher mittels kryptographischen Funktionen verhindert wird, während das Smart Device gesperrt ist und die Hintergrunddienste dennoch weitestgehend ungehindert lauffähig bleiben. Zu diesem Zweck partitioniert *Deadbolt* die Daten in zwei Bereiche. Der eine Bereich kann durch den Nutzer zur Laufzeit gesperrt, d. h. verschlüsselt werden. Dieser Bereich ist für alle Daten gedacht, die nicht zur Aufrechterhaltung der Funktionalität der mobilen Plattform und den Hintergrunddiensten benötigt werden. Der zweite Bereich bleibt dauerhaft unverschlüsselt. Zum Entsperren muss eine Personal Identification Number (PIN) eingegeben werden. Danach stehen dem Nutzer und seinen Apps alle Daten wieder zur Verfügung. Dieser Schutz richtet sich in erster Linie gegen physische Angriffe und nicht, wie der *SDC* respektive der *SDC+* gegen Angriffe von Apps zur Laufzeit.

In Systemen wie *Deadbolt* kann ein Nutzer seine Daten sichern, indem er sein Smart Device sperrt. Selbst wenn er sehr gewissenhaft mit der Sperrfunktion umgeht und diese immer nutzt, sobald er das Gerät aus der Hand gibt, kann es zu Situationen kommen, in denen dies nicht möglich ist. Beispielsweise müssen sogenannte Handy-Tickets, also Fahrkarten oder Eintrittskarten, die virtuell als Dokument auf einem Smart Device zugestellt werden, einem Kontrolleur jederzeit vorgezeigt werden können. Hierzu muss diesem das entsperrte Smart Device übergeben werden; der Kontrolleur hat in diesem Moment vollen Zugriff auf alle darauf gespeicherten Daten.

Die Plausibly Deniable Encryption (PDE) [CDNO97] zieht dies in Betracht. Ein Nutzer besitzt unter PDE mehrere Schlüssel, mit denen er seine Daten de-

chiffrieren kann. Allerdings führt nur einer dieser Schlüssel zu dem korrekten Klartext. Alle anderen Schlüssel ergeben einen Text, der für einen Angreifer korrekt erscheint (wenn es sich bei einem Datum beispielsweise um das Alter des Nutzers handelt, so erzeugt ein falscher Schlüssel einen Wert innerhalb eines realistischen Bereichs). Um Dritten dennoch eine Teilmenge der Daten, die diese lesen dürfen, zukommen lassen zu können, werden die Daten unterschiedlichen Sicherheitsstufen zugeteilt. Jeder Schlüssel entschlüsselt jeweils eine Untermenge dieser Stufen korrekt. Während für Desktop PCs mehrere PDE-Systeme existieren, haben PDE-basierte Lösungen für mobile Plattformen häufig viele Einschränkungen bezüglich der Performanz oder den von ihnen unterstützen Speichertypen. *MobiPluto* von Chang et al. [CWCZ15] ist eines der wenigen Systeme, das diese Einschränkungen nicht besitzt. Das Hauptproblem bleibt allerdings, dass der Nutzer sehr viele Schlüssel verwalten muss, um eine ausreichend große Menge an Sicherheitsstufen zu unterstützen und die Einteilung der Daten in die Sicherheitsstufen sehr aufwendig ist. Auch muss der Nutzer entscheiden können, welche Instanz Zugriff zu welcher Sicherheitsstufe haben sollte.

Die Untersuchungen von Davida, Wells und Kam [DWK81] befassen sich mit der Verschlüsselung von Datenbanken im Allgemeinen. Dabei zeigt sich, einerseits, dass es sicherer ist, wenn nicht einzelne Tupel verschlüsselt in der Datenbank vorliegen, sondern ein Angreifer von vornherein nicht in der Lage ist, auf die Datenbank zuzugreifen. Andererseits sollten mehrere Schlüssel zum Einsatz kommen, die jeweils nur einen Teil der Daten entschlüsseln – dies ist sowohl aus Performanz- als auch aus Sicherheitsgründen sinnvoll. Beides ist im *SDC* und dem *SDC+* von Grund auf berücksichtigt (siehe Abschnitt 5.2).

Bevor jedoch Anfragen auf dem *SDC* und dem *SDC+* ausgeführt werden können, müssen die Datenpartitionen zunächst vollständig entschlüsselt werden. Andere Ansätze zielen darauf ab, Datenbanken so zu verschlüsseln, dass einfache Anfragen bereits auf den chiffrierten Daten durchgeführt werden können und so eine Vorauswahl getroffen werden kann, damit anschließend nur relevante Datensätze entschlüsselt werden müssen. Dabei gibt es eine Vielzahl an unterschiedlichen Verfahren, wie beispielsweise *Searchable Symmetric Encryption* [CGKO06], *Fully-Homomorphic Encryption* [Gen09] oder *Order-Preserving Symmetric Encryption* [BCLO09]. Da das Ziel der Verschlüsselung allerdings

gerade darin besteht, Daten vor illegalen Zugriffen zu schützen, stehen diese Ansätze allesamt im Widerspruch zu Datensicherheit [AEKR14]. Lassen sich aus den verschlüsselten Datenbanken Rückschlüsse auf die darin gespeicherten Daten ziehen, so werden dadurch viele Angriffe ermöglicht [NKW15]. Trotz der möglicherweise besseren Performanz einer solchen Verschlüsselungstechnik, kommen diese für den *SDC* und den *SDC+* nicht in Frage, da dort die Datensicherheit im Fokus steht. Außerdem zeigen Messungen, dass die Performanz des *SDCs* und des *SDC+* auch ohne diese Verschlüsselungstechniken überzeugend ist (siehe Abschnitt 7.2).

5.5.2 Datenaustauschsysteme

Choe et al. [CBJP11] versuchen mit *MetaService* der Problematik des Datenaustauschs zwischen Apps unter Android Herr zu werden. Innerhalb einer App liegen die Daten in Form von Java-Objekten vor und können als solche direkt verarbeitet werden. *Content Provider* hingegen können nur auf strukturierten Datensätzen, wie beispielsweise Einträge einer Datenbank, arbeiten. Daher muss eine App ihre Daten in einer solchen Form ablegen, wenn sie sie mit anderen Apps teilen und austauschen will. Darüber hinaus verursacht bereits die Erstellung eines *Content Providers* einen erheblichen Arbeitsaufwand für den App-Entwickler. *MetaService* löst beide Probleme. Es agiert als Android-Service, der von jeder App genutzt werden kann. Als Interface bietet er in erster Linie zwei Funktionen an, mit denen beliebige Java-Objekte an ihn übergeben bzw. von ihm geholt werden können. Auf diese Weise wird zwar der Austausch von Daten erheblich erleichtert, allerdings kann der Zugriff auf den *MetaService* nicht reglementiert werden, d. h. jede App kann alle im *MetaService* abgelegten Objekte anfordern. Ein solcher Ansatz trägt daher weder zum Datenschutz noch zur Datensicherheit bei.

Ali-Gombe et al. [ARAR16] stellen mit *priVy* einen Ansatz vor, mit dem der Zugriff auf Datenbanken über *Content Provider* kontrolliert werden kann. Dabei kann für jeden Eintrag in der Datenbank angegeben werden, ob ein Zugriff erlaubt, mit Restriktionen erlaubt oder verboten ist. Wenn nötig schreibt *priVy* hierfür die Datenbankanfragen um, damit das Anfrageergebnis keine gesperrten Informationen enthält. Dadurch ist ein Datenaustausch mit einer feingranularen Berechtigungskontrolle möglich. Allerdings wird *priVy* in den

Bytecode jeder App eingefügt, wodurch der Ansatz gegen geltendes Recht verstößt (siehe Abschnitt 3.2). Außerdem stellt *priVy* keinen Schutz für die Datenbank selbst dar – erlangt ein Angreifer Zugriff auf diese, kann er alle darin gespeicherten Daten lesen. Auch ist nicht geklärt, wer die sehr umfangreichen und komplexen Berechtigungsregeln erstellt. Ein Nutzer müsste hierfür exakt wissen, welche Apps Datenbanken verwenden, wie deren Schemata sind und welche *Content Provider* existieren. Mutti, Bacis und Paraboschi [MBP15] wählen mit *SeSQLite* einen ähnlichen Ansatz, indem sie *SELinux* in die *SQLite*-Bibliothek integrieren. Dadurch kann auf allen Datenbankelementen Mandatory Access Control (MAC) angewandt werden. Durch diese Integration entfällt die rechtliche Problematik von *priVy*; die restlichen bleiben allerdings bestehen. Da *SELinux* für Mehrbenutzersysteme konzipiert ist, unterstützt *SeSQLite* theoretisch Nutzerkonten. Die Verwaltung der Konten muss allerdings von jeder App selbst implementiert werden.

5.5.3 Datenvernichter

Hat ein Angreifer physischen Zugriff auf ein Smart Device, so sind die privaten Daten eines Nutzers in Gefahr. Selbst wenn die Daten darauf verschlüsselt vorliegen, besteht die Gefahr, dass sich Kopien des Schlüssels aus dem Speicher auslesen lassen. Geambasu et al. [GJG+11] stellen daher mit *Keypad* einen Ansatz vor, mit dem die Daten im Falle eines Verlusts des Smart Devices unbrauchbar gemacht werden können. Auch *Keypad* verschlüsselt zu diesem Zweck alle privaten Daten, die Schlüsselverwaltung befindet sich allerdings nicht auf dem Gerät, sondern auf einem externen Server. Diesem kann ein Nutzer melden, wenn dessen Daten vernichtet werden müssen. Anschließend werden die zugehörigen Schlüssel vernichtet. Aber selbst wenn der Nutzer den Verlust des Geräts nicht bemerkt, schützt *Keypad* seine Daten. Immer wenn eine App beim Server einen Schlüssel anfragt, prüft dieser, ob die Anfrage zu dem Verwendungsmuster des Nutzers passt. Nur wenn keine Auffälligkeiten festgestellt werden, werden die Daten entschlüsselt. Allerdings sind hiervon immer alle Daten betroffen. Eine feingranulare Partitionierung ist nicht möglich.

Yu et al. [YWS+14] befassen sich mit der Problematik, dass eine Fernlöschung von Daten nur dann funktioniert, wenn ein Smart Device eine Internetverbindung besitzt und der Service das Gerät erreichen kann. Trennt ein

Angreifer daher rechtzeitig diese Verbindung, so kann die Löschung nicht erfolgen. Mit *Remote Wipe-Out* stellen sie daher ein Verfahren vor, das hierfür den Notrufmechanismus nutzt. Diese Verbindung bleibt selbst dann noch aktiv, wenn keine SIM-Karte vorhanden ist. Allerdings setzt *Remote Wipe-Out* das Gerät nur in den Werkszustand zurück. Dies stellt keinen zuverlässigen Schutz für die Daten dar, da forensische Verfahren sie wiederherstellen können.

Mit der unwiderruflichen Löschung von Daten befassen sich Diesburg et al. [DMS+16] in *TrueErase*. Das System unterstützt unterschiedliche Speichertypen, u. a. auch Flash-Speicher, wie er in Smart Devices zum Einsatz kommt. Hierfür verknüpfen sie jeden Datenblock auf dem Speicher mit Metadaten, die spezifizieren, welche Dateien diesen Block verwenden. Will der Nutzer eine spezifische Datei löschen, so kann *TrueErase* diese Hilfsstrukturen nutzen, um die zugehörigen Datenblöcke mehrfach zu überschreiben. Hierfür benötigt *TrueErase* vom Betriebssystem allerdings einen vollen und unbeschränkten Zugriff auf den Speicher. Unter Linux ist dies durch eine Erweiterung des Kernels möglich. Da Android auf Linux basiert, sollte sich *TrueErase* daher auch für diese mobile Plattform umsetzten lassen. Allerdings stellt *TrueErase* nur eine Ergänzung zu einem Datensicherheitssystem dar, da es Daten vor illegalen Zugriffen nicht schützen kann. Für den *SDC* und den *SDC+* kann das Verfahren dafür genutzt werden, um die Schlüssel sicher zu löschen. Auch kann *TrueErase* nur ganze Dateien löschen, wodurch die Löschgranularität sehr grob ist.

5.5.4 Datenpartitionierungssysteme

Samsung KNOX basiert auf der Android-Plattform, fügt aber in jeder Schicht der Systemarchitektur Sicherheitsfunktionen hinzu. Hierzu zählen u. a. ein sicherer Bootloader, Sicherheitserweiterungen im Linux Kernel, verbesserte Sandboxen und Verschlüsselungsfunktionen. Alle Erweiterungen sind vom Department of Defense (DoD) zertifiziert. Da *Samsung KNOX* in erster Linie auf Geschäftsanwendungen ausgerichtet ist, stellt die Einführung von getrennten Speicherbereichen in der Anwendungsschicht das wichtigste Feature dar. Apps aus dem einen Bereich (z. B. „private Apps“) können nicht auf Apps aus dem anderen Bereich (z. B. „geschäftliche Apps“) zugreifen. Dadurch kann das gleiche Smart Device in mehreren Domänen verwendet werden, ohne dass die jeweiligen Daten vermischt werden. Dies kann durch eine physische

(d. h. getrennte Speichermedien) oder logische Partitionierung (d. h. getrennte Adressbereiche) erzielt werden [Ent13].

Colp et al. [CZG+15] sehen in einer logischen Partitionierung keinen ausreichenden Schutz, da forensische Analysen diesen auslesen können, wenn ein Angreifer physischen Zugriff auf ein Smart Device erhält. Mit *Sentry* nutzen sie daher die Prozessorarchitektur heutiger Smart Devices aus. Ein Speicherbereich, der sich auf dem Prozessor befindet, ist speziell gegen physische Angriffe geschützt. Die Daten in diesem Speicher sind im ausgeschalteten Zustand vollständig verschlüsselt und sie werden erst beim Systemstart entschlüsselt. *Sentry* speichert private Daten in diesem Bereich. Allerdings ist dieser Speicher primär für die schnelle Ausführung von Code gedacht, weshalb die Menge der hierin speicherbaren Daten stark beschränkt ist. Auch schützt dieses Verfahren die Daten nicht gegen Zugriffe von anderen Apps oder gegen externe Zugriffe, so lange das Gerät eingeschaltet ist.

5.5.5 Nutzerverwaltungssysteme

Rohrer et al. [RZCZ13] stellen mit *DR BACA* einen dynamischen rollenbasierten Zugriffsmechanismus für Android vor. Hierbei werden Role Based Access Control (RBAC)-Techniken von klassischen Mehrbenutzersystemen mit Kontextdaten des Smart Devices verbunden, um dynamisch entscheiden zu können, ob ein Nutzer in einer bestimmten Situation Zugriff auf eine bestimmte Ressource erhalten darf. Neben dem reinen Zugriff kann ein Nutzer auch beschreiben, mit wem oder mit welcher App Berechtigungen geteilt werden dürfen. Regeln, die dies beschreiben, werden auf einem zentralen Server abgelegt und an alle Geräte, die *DR BACA* verwenden, gesendet. Dadurch können sich mehrere Nutzer ein Gerät teilen und für jeden gelten eigene Berechtigungen. Selbst wenn ein Nutzer das Gerät wechselt, werden seine Berechtigungen automatisch in diesem angewendet.

Während mit *DR BACA* Berechtigungen für Nutzer verwaltet werden können, stellen Nauman et al. [NKO+12] mit *POAuth* einen Autorisierungsmechanismus für mobile Plattformen vor. Es basiert auf *OAuth* [HAB+09] und erlaubt damit, Datenanbietern Zugriffsregeln für Datenkonsumenten zu definieren. Als Datenanbieter gilt in *POAuth* jede Instanz, die sensible Daten erzeugt. Datenkonsument kann jede beliebige App oder ein anderer Nutzer sein. *POAuth*

erweitert *OAuth* dahingehend, dass die Zugriffsregeln immer an ein Gerät gebunden sind, d. h. es findet eine Partitionierung auf Geräteebene statt. Berechtigungen lassen sich dadurch nicht heimlich auf fremde Geräte übertragen. Illegale Datenzugriffe werden durch das System allerdings nicht verhindert, da diese nicht geschützt sind (z. B. durch kryptographische Verfahren).

5.5.6 App-Scanner

Viele Anbieter von Antiviruslösungen für Desktop PCs bieten spezielle Versionen ihre Produkte auch für mobile Plattformen an. *Mobile Security* von *Trend Micro* ist eine der am häufigsten verwendeten Antivirus-Apps für Android. Neben einem reinen Virenschutz besitzt *Mobile Security* auch ein Reputationssystem, das Apps aufgrund ihres Umgangs mit privaten Daten bewertet, und dem Nutzer Empfehlungen zu deren Verwendung gibt. Über *Mobile Security* kann das Smart Device im Falle eines Diebstahls auch gesperrt oder auf den Werkszustand zurückgesetzt werden [Tre16]. Obwohl der Funktionsumfang dieser App zunächst überzeugt, zeigen Untersuchungen von Rastogi, Chen und Jiang [RCJ13], dass sehr viele schadhafte Apps nicht erkannt werden. Dies ist allerdings nicht nur ein Problem von *Mobile Security*, sondern betrifft alle App-Scanner, die auf einem Smart Device zur Laufzeit arbeiten [CY16]. Jarabek, Barrera und Aycock [JBA12] stellen daher mit *ThinAV* einen Cloud-basierten Virenscanner für mobile Plattformen vor. *ThinAV* überprüft Apps bei der Installation und verhindert diese, sollte es sich dabei um Schadsoftware handeln. Wird erst nach der Installation festgestellt, dass eine App schadhaft ist, so kann diese jederzeit von *ThinAV* gelöscht werden. App-Scanner benötigen in der Regel keine weiteren Schutzfunktionen; deren Ziel ist es, alle Schadprogramme zu stoppen, bevor diese aktiv werden können.

5.5.7 Zusammenfassung

Tabelle 5.2 bietet einen Überblick über die betrachteten verwandten Arbeiten. Als Referenzwerte sind zusätzlich Android sowie der *SDC* und der *SDC+* aufgeführt. Das Android-Berechtigungssystem wird in Kapitel 4 bewertet. Android verwendet FDE ausschließlich auf der Nutzerdatenpartition und der Schutz besteht nur, wenn das Gerät ausgeschaltet ist und nicht wenn es nur

	Feature							
System	Berechtigungen	Verschlüsselung	Datenaustausch	Löschfunktion	Partitionierung	Sicherheitsstufen	Nutzerkonten	App-Prüfung*
Android	(✓)	(✓)	(✓)	(✓)	✗	✗	✗	(✓)
AndrABEn	(✓)	✓	(✓)	✗	✗	✗	(✓)	✗
Deadbolt	✗	✓	✗	✗	✓	✗	✗	✗
MobiPluto	✗	✓	✗	✗	✗	✓	✗	✗
MetaService	✗	✗	(✓)	✗	✗	✗	✗	✗
priVy	✓	✗	✓	✗	✓	✗	✗	✗
SeSQLite	✓	✗	✓	✗	✓	✗	(✓)	✗
Keypad	(✓)	✓	✗	✓	✗	✗	✗	(✓)
Remote Wipe-Out	✗	✗	✗	(✓)	✗	✗	✗	✗
TrueErase	✗	✗	✗	✓	✗	✗	✗	✗
Samsung KNOX	✗	✓	✗	✗	✓	✓	✗	✗
Sentry	✗	(✓)	✗	✗	✓	✓	✗	✗
DR BACA	✓	✗	✓	✗	✗	✗	✓	✗
POAuth	✓	✗	✓	✗	(✓)	✗	✓	✗
Mobile Security	✗	✗	✗	(✓)	✗	✗	✗	✓
ThinAV	✗	✗	✗	✗	✗	✗	✗	✓
SDC	✓	✓	✓	✓	(✓)	✓	✗	✗
SDC+	✓	✓	✓	✓	✓	✓	✓	✗

* Untersuchungen zeigen, dass die App-Prüfung unter Android nahezu ineffektiv ist [CY16].

Tabelle 5.2: Vergleich von Sicherheitssystemen für mobile Plattformen

gesperrt wird. Zusätzlich zeigen Untersuchungen, dass das hierfür verwendete Verfahren von Angreifern ausgehebelt werden kann [SM16b]. Auch sind die Basiskryptografiealgorithmen von Android sehr unsicher [EBFK13]. Keiner der Mechanismen zum Datenaustausch, die Android anbietet, ist sicher und durch den Nutzer sinnvoll regulierbar [SDB14; OD16]. Meldet ein Nutzer sein Smart Device in seinem Google-Konto an, so kann er es darüber lediglich auf den Werkszustand zurücksetzten lassen. Daten lassen sich unter Android weder in Partitionen noch Sicherheitsstufen einteilen und eine Nutzerunterscheidung findet nicht statt, d. h. jeder Nutzer hat die gleichen Rechte [SS16]. Da die Android-Sperrmechanismen nicht sicher sind [SS12a], stellt dies eine große Bedrohung dar, sollte ein Angreifer physischen Zugriff auf ein Smart Device erlangen. Wird eine App über *Google Play* installiert, so kann diese von Google gelöscht werden, wenn Nutzer diese als Schadsoftware melden; eine automatische App-Prüfung findet jedoch nicht statt.

Wie man bei dem Vergleich sehen kann, besitzt keines der aktuell verfügbaren Sicherheitssysteme alle Eigenschaften des *SDCs* oder des *SDC+*. Mit diesen Eigenschaften lassen sich die in Abschnitt 1.3 aufgestellten Schutzziele [IS1] bis [IS9] erfüllen (siehe Abschnitt 7.1). Die Kombination dieser Eigenschaften kann daher als Alleinstellungsmerkmal des *SDCs* und des *SDC+* angesehen werden. Dass weder die *PMP* noch der *SDC* und der *SDC+* eine App-Prüfung unterstützen, ist aus zweierlei Gründen nicht kritisch: (1) App-Prüfungen auf mobilen Plattformen sind nahezu ineffektiv [CY16]. (2) Durch die restlichen Sicherheitsfeatures wird sichergestellt, dass eine bösartige App keinen Schaden anrichten kann.

Die Kombination aus *PMP* und *SDC* respektive *SDC+* stellt daher ein holistisches Sicherheitssystem dar. In Kapitel 6 werden vier unterschiedliche Anwendungsbeispiele für ein solches System gegeben, um zu demonstrieren, wie vielseitig die in der vorliegenden Arbeit vorgestellten Konzepte sind.

Kapitel 6

Anwendungsbeispiele für die PMP

Holistische Sicherheitssysteme können in einer Vielzahl von Anwendungsfällen Verwendung finden. Im Folgenden werden daher Praxisbeispiele aus vier unterschiedlichen Domänen herausgearbeitet. Anhand dieser Beispiele wird der Nutzen der *Privacy Management Platform (PMP)* demonstriert.

Dieses Kapitel basiert auf den eigenen Publikationen [Sta09], [BS09], [Sta11], [SB11] (Anwendungsfall „Ortsbasierte Anwendungen“ in Abschnitt 6.1), [SS12b], [Sta16], [GS17] (Anwendungsfall „mHealth-Anwendungen“ in Abschnitt 6.2), [GS14], [GSMW16] (Anwendungsfall „Industrie 4.0“ in Abschnitt 6.3) sowie [CSDM12] (Anwendungsfall „Internet der Dinge“ in Abschnitt 6.4).

6.1 Ortsbasierte Anwendungen

Mit dem Aufkommen der Smart Devices gewannen auch ortsbasierte Anwendungen immer mehr an Bedeutung [ZN06]. Auch wenn es für den Begriff „ortsbasierte Anwendung“ keine scharfe Definition gibt, werden in der Regel

darunter alle Dienste zusammengefasst, die den Kontext des Nutzers primär aus dessen Standortdaten ableiten und sich darauf anpassen [Küp05]. Diese Sonderstellung des Datums „Standort“ ist nur wenig verwunderlich, da der Nutzer einen immensen Mehrwert durch Einbeziehung seiner Standortdaten erfährt [LK06]. Untersuchungen von Burak und Sharon [BS03] haben hingegen gezeigt, dass der Nutzer im Umgang mit seinen Standortdaten in der Theorie sehr kritisch ist und die Anwendungen lieber anonym verwenden möchte. In der Praxis kommen jedoch selten Datenschutzmechanismen zum Einsatz, wenn diese die Servicequalität reduzieren oder zu kompliziert zu bedienen sind [BS04].

Im Rahmen der eigenen Vorarbeiten wurden zwei Kategorien von ortsbasierten Anwendungen besonders berücksichtigt: ortsbasierte Spiele und Mitfahrdienste. In ortsbasierten Spielen steuert der Spieler seinen Avatar dadurch, dass er sich in der realen Welt bewegt. Dabei kann es sich um Mikrobewegung, d. h. Veränderungen der Ausrichtung des Smart Devices, handeln oder um Positionsveränderungen des Spielers. Ersteres kann beispielsweise über einen Beschleunigungs- oder Orientierungssensor erfasst werden. Dadurch werden nur wenige Informationen über den Nutzer mit dem Spiel geteilt. Für letzteres werden allerdings über den Verlauf des Spiels Standortdaten in Echtzeit erfasst (z. B. via Global Positioning System (GPS)). Dadurch hat das Spiel Zugriff auf sensitive Daten und gefährdet die Privatsphäre des Nutzers.

In den eigenen Publikationen [Sta09] und [BS09] wird ein mobiles ortsbasiertes Spiel namens *Pervasive Tic Tac Toe* vorgestellt und es wird untersucht, wie hierin der Standort des Nutzers einbezogen wird. Das Spiel wurde im Rahmen der eigenen Vorarbeit [Sta09] implementiert. In *Pervasive Tic Tac Toe* spielen zwei Spieler eine Partie *Tic Tac Toe* [Do05] auf Smart Devices gegeneinander. Im Gegensatz zu der normalen Papierversion des Spiels, wird in *Pervasive Tic Tac Toe* das Spielfeld auf einem Landkartenausschnitt angezeigt. Der gewählte Ausschnitt liegt nahe dem gegenwärtigen Aufenthaltsort des Spielers. Um ein *Tic-Tac-Toe*-Feld zu besetzen, muss sich der Spieler an Koordinaten bewegen, die mit diesem Feld verknüpft sind. Damit das Spiel feststellen kann, wo sich der Spieler aufhält, werden dessen GPS-Daten dauerhaft benötigt.

Genau wie in nahezu allen ortsbasierten Spielen ist bei *Pervasive Tic Tac Toe* der Standort das einzige schützenswerte Datum. Die *PMP* könnte hier gewinn-

bringend eingesetzt werden. Die meisten Datenschutzsysteme ermöglichen es einem Nutzer nur, einer App entweder den Zugriff auf die Standortdaten vollständig zu verbieten oder diese mit komplett verfälschten Werten zu versorgen (siehe Abschnitt 4.5). Dadurch würde das Spiel allerdings unspielbar werden. Im Gegensatz dazu kann mit einer *PMP*-GPS-*Resource* die Genauigkeit der Standortdaten reduziert werden. Dies hätte zwar Auswirkungen auf das Spiel (es kann nicht mehr exakt ermittelt werden, an welcher Position sich der Nutzer befindet), aber das Spiel wäre auch weiterhin spielbar. In einer erweiterten Version der GPS-*Resource*, könnte man über zusätzliche *Privacy Settings* ermöglichen, dass mit dem Spiel nur relative Positionsveränderungen anstelle absoluter Positionsdaten geteilt werden. Auf diese Weise könnte das Spiel ausgehend von einer fiktiven Startposition genau ermitteln, wo sich ein Spieler auf dem Spielfeld befindet, ohne jemals dessen reale Position zu kennen. Die *PMP* könnte so für eine Vielzahl an ortsbasierten Spielen die privaten Daten des Nutzers vollständig schützen, ohne die Servicequalität der Spiele einzuschränken.

Neben Spielen stellen vor allem Mitfahrdienste einen Großteil der ortsbasierten Dienste dar. Ein Mitfahrdienst richtet ein soziales Netzwerk ein, das das Ziel hat, potentielle Fahrer und Beifahrer zusammenzubringen [GRB10]. Die Planung funktioniert umso besser, je mehr Kontextdaten zu einem Nutzer und dessen Fahrten dem Netzwerk zur Verfügung stehen (z. B. dessen aktueller Standort oder dessen Vorlieben) [SKA+10]. Viele dieser Dienste setzen ein Online-Netzwerk auf und planen darüber die Fahrten [CML15]. Dies bedeutet aber, dass ein Nutzer sämtliche seiner Daten mit dem Dienst teilt. Besonders unter Datenschutzaspekten sind daher Ad-hoc-Mitfahrdienste besser, bei denen die Teilnehmer, bzw. deren Smart Devices, direkt miteinander kommunizieren und so automatisch passende Fahrer respektive Beifahrer finden [MBM12]. Auf diese Weise reduziert sich die Menge an Daten, die mit dem Netzwerk geteilt werden müssen. Nichtsdestotrotz müssen Kontextdaten, wie die Standorte der Nutzer, mit anderen Nutzern geteilt werden. Um dies reglementieren zu können, existieren Insellösungen, wie die Arbeit von Goel, Kulik und Ramamohanarao [GKR16]. Da das Datenschutzsystem dabei unmittelbar in den Mitfahrdienst integriert ist, muss man allerdings darauf vertrauen, dass sich der

Dienst, vor dem man die privaten Daten verheimlichen will, selbst zuverlässig reguliert.

In den eigenen Publikationen [Sta11] und [SB11] wird mit *vHike* ein Ad-hoc-Mitfahrdienst vorgestellt (siehe Anhang Abbildung B.1). *vHike* sucht potentielle (Bei-)Fahrer in der Umgebung des Nutzers via Bluetooth. Bei der Mitfahrervermittlung werden neben dem Standort auch geplante Fahrtrouten, durchgeführte Fahrten, die in der Vergangenheit liegen, oder Vorlieben, wie beispielsweise eine maximale Anzahl an Mitfahrern, berücksichtigt. Diese Vermittlung wird direkt zwischen den Smart Devices durchgeführt. Einzig Nutzerprofile werden auf einem zentralen Server hinterlegt. Um die Unversehrtheit der Nutzer sicherzustellen, können Fahrten optional auf dem zentralen Server abgespeichert werden, wodurch im Nachhinein nachvollzogen werden kann, wer wann mit wem wohin gefahren ist. Ein initialer Prototyp von *vHike* wurde im Rahmen des Softwarepraktikums von Steeb und Kiesewetter [SK10] erstellt.

Auch wenn in *vHike* ein Großteil der privaten Daten nicht über den zentralen Server laufen, werden diese dennoch mit der App und sogar mit anderen Nutzern geteilt. Daher könnte die *PMP* hier gewinnbringend eingesetzt werden. Mit restriktiven Datenschutzsystemen, die einer App die Verwendung von privaten Daten entweder verbieten oder diese mit vollständig verfälschten Werten versorgen, würde *vHike* unnutzbar sein. Die *PMP* könnte beispielsweise eine *vHike-Resource* zur Verfügung stellen, die überprüft, welche *vHike*-Nutzer sich in der näheren Umgebung befinden, ohne deren Position der App mitzuteilen. Hierfür würden die Standortdaten der Nutzer verschlüsselt ausgetauscht, und nur die *vHike-Resource* besitzt den Schlüssel. Diese *Resource* könnte alle nötigen Berechnungen durchführen und nur die anonymisierten Ergebnisse werden mit *vHike* geteilt. Auf gleiche Weise können Nutzerprofile mit den Vorlieben ausgetauscht werden, die lediglich von der *PMP* ausgewertet werden. Dies beeinflusst die Servicequalität von *vHike* nicht, aber weder die App noch andere Nutzer erhalten Zugriff auf private Daten. Eine Implementierung von *vHike* mit *PMP*-Unterstützung wurde im Rahmen des Studienprojekts von Vetter et al. [VJB+12] realisiert und durch das Projekt-INF von Braunbeck, Butz und Kleine [BBK14] erweitert.

Aber auch auf dem zentralen Server können die im Rahmen der vorliegenden Arbeit gewonnenen Erkenntnisse genutzt werden. Zur Speicherung der

Nutzerprofile sowie der abgespeicherten Fahrten kann eine spezielle Version des *Secure Data Container Plus (SDC+)* zum Einsatz kommen. Dabei werden dessen Sicherheitsmechanismen auf die Datenbank des Servers übertragen. Auf diese Weise hätten die Nutzer die Gewissheit, dass sie über den *Kill Switch* des *SDC+* ihr Profil auf Wunsch sicher und nachhaltig löschen können und dass der Betreiber des Servers keinen Zugriff auf ihre gespeicherten privaten Daten erhält. Eine solche Implementierung des *SDC+* für die Cloud wurde prototypisch in der Diplomarbeit von Mayer [May15] umgesetzt.

6.2 mHealth-Anwendungen

Während bei ortsbasierten Anwendungen im Wesentlichen die Standortdaten eines Nutzers von potentiellen Angreifern in Erfahrung gebracht werden können, womit sich u. a. ein Bewegungsprofil des Nutzers erstellen lässt, gibt es andere Anwendungen, die wesentlich sensiblere Daten verarbeiten. Ein Beispiel hierfür sind medizinische Anwendungen. Aufgrund von rapide steigenden Kosten im Gesundheitswesen müssen neue kostengünstige Behandlungsmethoden gefunden werden. Insbesondere bei chronischen Krankheiten, wie beispielsweise Diabetes, die dauerhaft eine regelmäßige medizinische Überwachung und Behandlung bedürfen, lohnt sich der Einsatz von Telemedizin [SRdlT+15]. Heutzutage wird hierbei für die Diagnose, Untersuchung und Therapie verstärkt auf *mHealth*-Technologien gesetzt, d. h. auf Apps für Smart Devices, die mit medizinischen Geräten verbunden werden können. Solche *mHealth*-Anwendungen lassen sich für alle Patientengruppen und Krankheiten bzw. Lebensumstände anpassen [Sie12]. Studien von Knöll [Knö10b] belegen, dass besonders junge Patienten über *mHealth*-Anwendungen angesprochen werden und ihnen so eine richtige, d. h. gesunde Lebensweise vermittelt werden kann. Auch lassen sich beispielsweise über Spiele therapeutische Maßnahmen in den Alltag von Kindern einbauen [KWKH14].

In der eigenen Publikation [SS12b] wird mit *Candy Castle (CC)* ein *mHealth*-Spiel für diabeteskranke Kinder vorgestellt (siehe Anhang Abbildung B.2). Das Spiel basiert auf einer Grundidee von Knöll [Knö10a], in der Kinder ermutigt werden regelmäßig ihren Blutzucker zu messen und diesen als Eingabewert für ein Spiel zu verwenden. Dabei wird der Blutzucker mit den Standortdaten, an

dem die Messung durchgeführt wurde, verknüpft, um später Erkenntnisse über die Einflüsse von bestimmten Orten auf die Gesundheit von Diabetespatienten bestimmen zu können. In *CC* ist diese Idee dadurch realisiert, dass der Spieler an seinem Wohnort ein virtuelles Schloss errichtet, das seine Gesundheit repräsentiert. Eine dunkle Macht (die Diabetes) greift dieses Schloss regelmäßig an. Daher muss der Spieler Mauern aufstellen, die diese Angriffe abwehren. Immer wenn der Spieler einen gemessenen Blutzuckerwert einträgt wird an dem Ort der Messung ein Mauerfragment erstellt. Auf diese Weise sind die Kinder motiviert an möglichst vielen Orten ihren Blutzucker zu überprüfen, und sie müssen regelmäßig an die Messungen denken, damit die dunkle Macht die Mauern nicht zerstört. Eine erweiterte Implementierung von *CC* für Android wurde im Rahmen der Bachelorarbeit von Jullien [Jul13] durchgeführt.

CC sieht sich genau wie alle *mHealth*-Anwendungen mit zwei schwerwiegenden Herausforderungen konfrontiert. Einerseits stellt die *Interoperabilität* von Smart Devices mit medizinischen Geräten häufig ein Problem für *mHealth*-Anwendungen dar [CEF+12]. Auch wenn die meisten medizinischen Geräte heutzutage in der Lage sind, Daten via Bluetooth austauschen zu können, gibt es kein einheitliches Übertragungsprotokoll. Einige Geräte verwenden den ISO/IEEE 11073 Standard für Gesundheitsdaten [ISO14] während andere auf proprietäre Protokolle, wie das *Terminal I/O* Protokoll [Sto15], setzen. Mit *Android Wear* [And16b] führt Google eine Methode zur Verbindung von Android-basierten Smartphones mit *Wearables*, d. h. Smart Devices, die am Körper getragen werden, ein. *Android Wear* wird allerdings in erster Linie von Smart Devices für den Massenmarkt, wie beispielsweise Smartwatches, und weniger von medizinischen Spezialgeräten unterstützt [RPP15]. Während es mehrere Arbeiten zur Vereinheitlichung des Datenmodells von Gesundheitsdaten gibt [OH12], ist für ein einheitliches Übertragungsprotokoll keine Lösung in Aussicht. Daher müssen mHealth-Anwendungen entweder eine Vielzahl an Übertragungsprotokollen unterstützen oder sich auf einen konkreten Gerätetyp festlegen, was ihre Nutzbarkeit jedoch stark einschränkt [JM11].

Andererseits muss die Datensicherheit [BDL14] und in besonderem Maße auch der Datenschutz [ABK12] sichergestellt werden. Einem Patient gegenüber muss gewährleistet werden, dass nur autorisierte Instanzen (z. B. Ärzte, aber auch spezielle Apps, die die Daten weiterverarbeiten) Zugriff auf seine

Gesundheitsdaten erhalten. Auch muss garantiert werden, dass diese Daten nicht verfälscht werden können. Gerade fehlendes Vertrauen in die Datensicherheit von *mHealth*-Anwendungen schrecken Nutzer vor deren Verwendung ab, obschon sie von ihnen profitieren könnten [MHK15].

Für beide Herausforderungen kann die *PMP* eine Lösung darstellen. Das Problem der Interoperabilität kann über die erweiterbaren *Resource Groups* gelöst werden. Zu jedem Übertragungsprotokoll kann in einer einzigen *Resource Group* für die Verbindung zu medizinischen Geräten eine eigene *Resource* angelegt werden. Ein App-Entwickler kann dadurch diese *Resource Group* verwenden und auf medizinische Daten zugreifen, ohne sich darum zu kümmern, von welchem Gerät sie stammen. Vergleichbar mit dem Konzept von Sorber et al. [SSP+12] entsteht so eine *Hub-and-Spoke*-Architektur mit der *PMP* als zentralen Knoten. Da *Resource Groups* und ihre *Resources* zur Laufzeit installiert oder aktualisiert werden können, kann ein Nutzer bei Bedarf jederzeit neue Übertragungsprotokolle nachrüsten [SSF17]. Genau wie es Brutschy et al. [BFTP15] vorschlagen, passen sich Apps dadurch automatisch an die verfügbare Hardware an.

Neben den bereits in Kapitel 4 und Kapitel 5 beschriebenen Datenschutz- und Datensicherheitsfeatures, kann die *PMP* zum Schutz von besonders sensiblen Daten mit einer speziellen Versiegelungs-*Resource* beitragen. Wird diese *Resource* verwendet, tauschen sämtliche *Resources* ihre Daten nur verschlüsselt mit anfragenden Apps aus. Dadurch können die Daten von der App nicht gelesen, aber an andere *Resources* weitergeleitet werden. Die Versiegelungs-*Resource* entschlüsselt die Daten ausschließlich für andere *Resources*, die diese verarbeiten. Auf diese Weise kann eine App beispielsweise mit stark gefilterten oder auch verfälschten Daten versorgt werden, während die *Resources* auf den korrekten Daten arbeiten. Darüber hinaus ist durch die Verschlüsselung sichergestellt, dass keine App sensible Daten manipulieren kann. Durch dieses an die Arbeit von Weerasinghe, Rajarajan und Rakocevic [WRR09] angelehntes Vorgehen, bleibt dem Nutzer unter Wahrung der Datensicherheit und des Datenschutzes die volle Servicequalität erhalten.

In den eigenen Publikationen [Sta16] sowie [GS17] wird basierend auf diesen Erkenntnissen *CC* erweitert und auf die *PMP* angepasst. Das Ergebnis wird als *Secure Candy Castle (SCC)* bezeichnet. Das Modell von *SCC* ist in

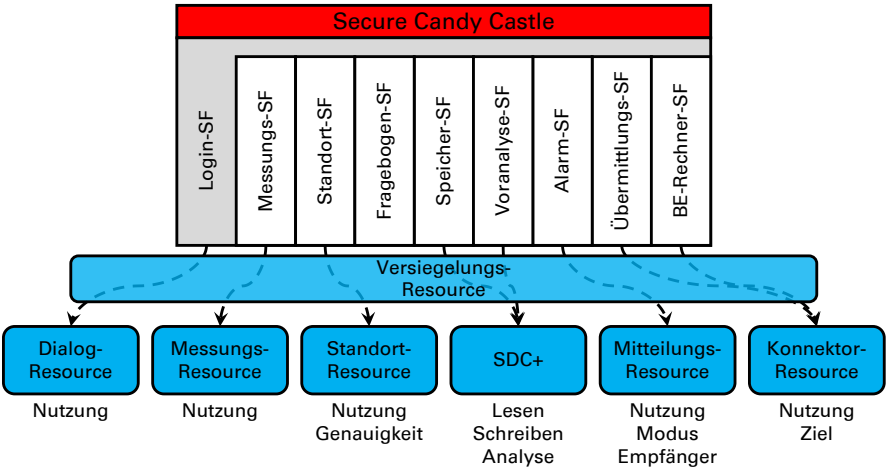


Abbildung 6.1: *Service Features* (*SF*) und *Resource Groups* von *SCC* (in Anlehnung an [Sta16])

Abbildung 6.1 dargestellt. *SCC* kann unmittelbar sechs standardisierte *Resources* nutzen, wodurch die Realisierung der Programmlogik erheblich erleichtert wird.

Die Dialog-*Resource* führt einen sicheren Login durch, ohne dass die Anmeldeinformationen mit *SCC* geteilt werden (analog zu der Login-Funktion vom *SDC+*, siehe Abschnitt 5.4). Die Messungs-*Resource* stellt die Verbindung zu einem Blutzuckermessgerät dar, und die Standort-*Resource* ermöglicht den Zugriff auf GPS-Daten. Im *SDC+* können die Gesundheits- und Spieldaten gespeichert und analysiert werden. Falls bei einer Analyse der Gesundheitsdaten eine kritische Situation ermittelt wird, können autorisierte Personen direkt per Mail oder Short Message Service (SMS) über die Mitteilungs-*Resource* informiert werden. Für die langfristige Speicherung und ausführliche Analyse können die Daten auf einem externen Server, wie beispielsweise dem *ECHO*-Back-End [BKK+15; SWM+15] über die Konnektor-*Resource* ausgelagert werden. Bei all diesen *Resources* kann der Nutzer über *Privacy Settings* regeln, ob *SCC* diese nutzen darf, und für einzelne *Resources* beispielsweise festlegen, wie genau diese die Daten an die App weitergibt oder an welches Ziel Daten von der App verschickt werden dürfen.

Damit *SCC* auch funktioniert, wenn ein Nutzer der Verwendung einer dieser *Resources* nicht zustimmt, ist die Programmlogik in einzelne *Service Features* aufgeteilt, die sich individuell aktivieren lassen. Zwei dieser *Service Features* sind herauszuheben. Das Login-*Service-Feature* kann nicht deaktiviert werden, da die Identifikation des Nutzers für die Verwendung von *SCC* erforderlich ist. Das Fragebogen-*Service-Feature* benötigt keinen Zugriff auf *Resources*. Damit kann der Nutzer seine Stimmung, seine Aktivitäten oder die Menge der Broteinheiten (BEs), die er zu sich genommen hat, festhalten; dies ist für Diabetespatienten sinnvoll [Pay15]. Obwohl hierfür keine *Resources* nötig sind, kann der Nutzer dennoch frei entscheiden, ob er diesen Fragebogen in *SCC* verwenden möchte. Der Datenaustausch mit den *Resources* läuft über die Versiegelungs-*Resource*, wodurch *SCC* selbst keinen Zugriff auf die Gesundheitsdaten erhält. Eine prototypische Implementierung von *SCC* wurde in der Bachelorarbeit von Giebler [Gie16b] durchgeführt und in dem Praktikum von Giebler [Gie16a] um weitere Funktionen ergänzt.

6.3 Industrie 4.0

Auch im industriellen Umfeld kommen immer mehr Smart Devices zum Einsatz [KLW11]. Im Zuge der vierten industriellen Revolution werden diese Geräte herangezogen, damit sie autonom Entscheidungen treffen, um so die Automatisierung der Industrie weiter voranzutreiben [Bau14]. Der Mensch wird dabei jedoch nicht außen vor gelassen; vielmehr wird seine Arbeit durch die Smart Devices unterstützt und somit seine Produktivität gesteigert [Hir14].

Gerade der Einsatz von Smart Devices und Apps verspricht hierbei ein enormes Leistungsvermögen, da die Arbeiter diese Geräte jederzeit mit sich führen können und sich die Apps jederzeit an den Kontext der Nutzer anpassen können [HGKM14]. Das Potenzial und die Einsatzmöglichkeiten von Apps in Bereich Industrie 4.0 wird jedoch gerade erst erforscht [HGKM15]. Allerdings steht der Einsatz solcher Apps auch großen Herausforderungen gegenüber, allem voran die Datensicherheit [ES12] sowie der flexible und feingranulare Zugriff auf relevante Geschäfts- und Prozessdaten [HGM15].

Im Rahmen der eigenen Vorarbeiten wurde daher untersucht, inwiefern ein Datenschutzsystem wie die *PMP* in einem industriellen Umfeld eingesetzt

werden kann und welche Anwendungsfälle sich in diesem Kontext ergeben. In der eigenen Publikation [GSMW16] wird analysiert, wie ein System beschaffen sein muss, das Fertigungsprozesse in Echtzeit überwacht und die Arbeiter mit Handlungsempfehlungen zur Erfüllung der Prozessziele versorgt. In einem solchen System sind Smart Devices unverzichtbar, da sie einerseits als Informationsanzeige für die Arbeiter und andererseits als Informationsquelle für das System zum Einsatz kommen. Über die in einem Smart Device verbauten Sensoren kann das Analysesystem wesentlich exakter bestimmen, wie die aktuelle Situation eines Arbeiters ist, und diesen somit mit präziseren Empfehlungen und Anweisungen versorgen. Auch kann die Rechenleistung der Smart Devices für eine Vorverarbeitung der Sensordaten genutzt und dem Back-End des Analysesystems aufbereitete höherwertige Daten bereitgestellt werden, um dieses zu entlasten. Als Prototyp für eine App im Fertigungsbereich wird das *Mobile Manufacturing Dashboard (MMD)* in der eigenen Publikation [GS14] vorgestellt (siehe Anhang Abbildung B.3).

Mit dem *MMD* wird das Ziel verfolgt, den Nutzer an seinem Arbeitsplatz zu jedem Zeitpunkt mit allen benötigten Informationen über den Prozess, den er aktuell ausführt, zu versorgen. Zu diesem Zweck werden von dem *MMD* die Sensoren seines Smart Devices, wie beispielsweise die Kamera, das Mikrofon oder das GPS, ausgelesen, um dessen aktuellen Kontext zu ermitteln. Um das *MMD* nutzen zu können, muss sich der Arbeiter gegenüber dem System authentifizieren. Hierbei wird ihm eine Rolle, die er im Rahmen des Prozesses einnimmt, zugewiesen. Diese Rolleninformation wird zusammen mit den Kontextdaten dafür verwendet, möglichst exakt bestimmen zu können, welche Informationen auf welche Weise dem Arbeiter angezeigt werden sollen. Um weitere Prozessdaten in Echtzeit im *MMD* verwenden zu können, muss das Smart Device mit den restlichen Maschinen in der Fabrik kommunizieren können. Für eine langfristige Analyse mit dem Ziel der Prozessoptimierung, übermittelt das *MMD* seine Daten an ein Analyse-Back-End, wie beispielsweise die *Advanced-Manufacturing-Analytics*-Plattform [GNSM12; Grö15]. Eine prototypische Implementierung des *MMDs* wurde in der Bachelorarbeit von Schneider [Sch13a] durchgeführt.

Da das *MMD* sowohl mit Geschäfts- bzw. Prozessdaten als auch mit sensiblen Nutzerdaten umgeht, ist unmittelbar ersichtlich, dass es von einem

Datensicherheits- und Datenschutzsystem profitiert. Einerseits kann die *PMP* die Authentifizierung über ihre Login-*Resource* universell für alle vergleichbaren Apps durchführen. Dadurch erhält die jeweilige App keinen Zugriff auf die Anmeldeinformation. Besonders für Apps, die von Drittanbietern zugekauft werden, ist dies ein entscheidender Vorteil. Da die Prozessdaten teilweise auch auf dem Smart Device zur Verarbeitung gespeichert werden müssen, kann hierfür der *Secure Data Container (SDC)* zum Einsatz kommen. Außerdem kann die *PMP* auf „fehlende" Sensoren über ihre flexiblen *Resources* reagieren und das *MMD* dynamisch über andere Sensoren mit den benötigten Informationen versorgen.

Aber auch der Arbeiter möchte seine Persönlichkeitsrechte gewahrt wissen. So ist das *MMD* beispielsweise in der Lage exakt zu erfassen, wie lange sich ein Arbeiter im Raucherraum aufhält, oder Gespräche zwischen Arbeitern aufzuzeichnen. Hierfür können die *Privacy Settings* der *PMP* genutzt werden, um die Sensordaten vor der Verarbeitung zu filtern oder zu anonymisieren. Beispielsweise kann eine *Resource* die Informationen zu *n* potentiellen Standorten respektive Prozessschritten anfordern und selbstständig die richtigen auswählen, so dass dem Back-End nicht bekannt ist, welcher Tätigkeit der Arbeiter aktuell exakt nachgeht; für den Arbeiter bleibt die Servicequalität der App allerdings unverändert. Bei dem Teil der Analyse, der auf dem Smart Device abläuft, kann eine Versiegelungs-*Resource* zum Einsatz kommen. Dadurch können innerhalb des *SDC+* die Daten unverfälscht liegen, wodurch die Analyseergebnisse korrekt sind. Die App wird von der Versiegelungs-*Resource* aber nur über relevante Ereignisse informiert, die keine Rückschlüsse auf private Daten des Arbeiters zulassen. Die externe Datenanalyse im Back-End liegt außerhalb des Fokus der vorliegenden Arbeit. Daher sei an dieser Stelle auf die Arbeit von Dwork und Nissim [DN04] verwiesen, die sich mit dieser Fragestellung befasst.

Ein weiteres großes Thema im industriellen Umfeld stellt Bring Your Own Device (BYOD) dar, also die Verwendung von privater Hardware (beispielsweise Smart Devices) für dienstliche Zwecke. BYOD kann einem Unternehmen viel Geld bei der Hardwarebeschaffung einsparen, und die Nutzer müssen sich nicht erst an den Umgang mit dem System gewöhnen. Besonders im Bezug auf Kleingeräte, wie beispielsweise Smartphones, kann BYOD daher wirtschaftliche

Vorteile bringen [FGS14]. Da diese Geräte allerdings einerseits nahezu immer eine Verbindung zum Internet besitzen und andererseits im Rahmen von BYOD in Kontakt mit geheimen Geschäftsdaten kommen, stellen sie ebenfalls eine große Bedrohung dar [WMM13]. Daher ist es nötig, dass die Unternehmen ihre Sicherheitsrichtlinien auf dieses neue Anwendungsfeld anpassen [VA15]. Jedoch halten viele Nutzer diese Sicherheitsrichtlinien nicht ein – sei es, dass sie Apps verwenden, die dagegen verstoßen, oder dass sie nicht die nötigen Sicherheitsstandards auf ihren Smart Devices einhalten [Tho12]. Die Hauptherausforderung besteht daher darin, dass sensible Daten des Unternehmens gegen Angriffe oder illegale Verarbeitung abgesichert werden, potentielle Malware keinen Schaden anrichten kann und die Sicherheitsrichtlinien durchgesetzt werden [CHC14; KW16]. Neben all diesen Datensicherheitsproblemen kommen aus Nutzersicht noch Datenschutzbedenken hinzu – neben den Sensordaten könnten auch die Daten des Nutzers, die auf dessen privatem Smart Device gespeichert sind, preisgegeben werden [MVH12].

Die *PMP* kann auch zur Lösung dieser Herausforderungen beitragen und so zum Enabler für BYOD werden. Die Administratoren können die Sicherheitsrichtlinien eines Unternehmens in *Privacy Rules* für die *PMP* formulieren. Diesen *Privacy Rules* kann ein Kontext hinzugefügt werden (z. B. der Standort und die Zeit), der die Arbeitszeiten des Unternehmens eindeutig beschreibt. Über die *Presets* können diese *Privacy Rules* an die Angestellten verteilt werden. Panos [Pan13] untersuchte im Rahmen seiner Diplomarbeit, wie sichergestellt werden kann, dass jeder Nutzer diese *Presets* bei sich installiert und er diese nicht nachträglich verändern kann. Hierzu wurde der *PMP* ein Service hinzugefügt, der während der definierten Arbeitszeit regelmäßig die *Privacy Policy* des Nutzers mit dem *Presets* des Unternehmens abgleicht, welches jederzeit von einem zentralen Server abgerufen werden kann. Auf diese Weise werden Manipulationen des Nutzers zeitnah festgestellt und Änderungen an den Sicherheitsrichtlinien direkt übernommen.

Zusätzlich kann das Unternehmen *Resources* erstellen, über die sicher und kontrolliert auf das Unternehmens-Back-End zugegriffen werden kann. Dadurch können bereits clientseitig Zugriffsrechte auf Unternehmensdaten überprüft und mögliche Angriffe abgewehrt werden. Darüber hinaus kann über die *Privacy Settings* der Internetzugriff während der Arbeitszeit blockiert oder ein-

geschränkt werden, um so sicherzustellen, dass keine Unternehmensdaten an externe Parteien gesendet werden. Es kann auch der Zugriff auf den internen Speicher dahingehend reglementiert werden, dass keine sensiblen Informationen dauerhaft darauf gespeichert werden können, da diese sonst vom Nutzer ebenfalls an Dritte weitergegeben werden können. Müssen dennoch Daten dauerhaft auf dem Smart Device gespeichert werden, so kann die Partitionierung des *SDC+* genutzt werden. In Analogie zu *Samsung KNOX* [Ent13] können auf diese Weise private Daten und Unternehmensdaten strikt getrennt werden. Auch kann über den *SDC+* sichergestellt werden, dass der Nutzer außerhalb der Arbeitszeit keinen Zugriff auf die Unternehmenspartition erhält.

6.4 Internet der Dinge

Damit Industrie 4.0 Wirklichkeit werden kann und Apps wie das *MMD* realisiert werden können, genügt es nicht, Nutzer mit Smart Devices auszustatten. Hierfür müssen Sensoren unmittelbar an den Maschinen angebracht sein, die am Fertigungsprozess beteiligt sind [eur16]. Zusätzlich müssen auch die Maschinen untereinander vernetzt sein und autonom miteinander kommunizieren können, um sich einen Überblick über den gesamten Prozess verschaffen zu können [WAX15]. Wird ein Gerät um Technologien erweitert, die es ihm ermöglichen, seine Umgebung wahrnehmen zu können und diese Daten über die Umgebung mit anderen Geräten auszutauschen, so spricht man allgemein von *Smart Things* [AAS13].

Im Gegensatz zu Smart Devices stellen Smart Things diese Technologien dem Nutzer allerdings nicht unmittelbar zur Verfügung; Smart Things agieren in einer sogenannten *Smart Environment* weitestgehend selbstständig miteinander [FV13]. Beispiele für solche Smart Environments sind u. a. *Smart Homes*[1] oder *Smart Factories*[2]. Da die Smart Things und nicht der Mensch in einer Smart Environment sowohl Informationsquelle als auch -senke sind, spricht

[1] Von einem Smart Home spricht man, wenn Haushaltsgeräte und Unterhaltungselektronikgeräte miteinander verknüpft werden [GBMP13]. Ein Beispiel hierfür wäre ein Haus, bei dem sich die Jalousien in der Dämmerung automatisch schließen und gleichzeitig die Beleuchtung eingeschaltet wird.

[2] Eine Smart Factory ist eine Produktionsumgebung, in der sich cyber-physische Systeme überwiegend autonom organisieren [Jaz14]. In Franco da Silva et al. [FBK+16] ist ein Beispiel für ein temperaturgesteuertes Kühlsystem gegeben.

man hierbei allgemein vom Internet der Dinge respektive Internet of Things (IoT) [MWC13].

Im Bereich des Internets der Dinge liegt eine große Herausforderung in der Verarbeitung der Daten, da jedes einzelne Smart Thing nur über eine eingeschränkte Rechenleistung verfügt. Daher wird in jeder Smart Environment eine zentrale Komponente benötigt, die die Verarbeitung übernimmt. Diese leitet aus den Sensordaten höherwertige Ereignisse ab und informiert die in der Smart Environment installierten Smart Things darüber, damit diese darauf ggf. reagieren können [FV13]. Hierbei können Complex Event Processing (CEP)-Techniken [Luc01; BD15] zum Einsatz kommen. *SitRS* [HWS+15] respektive deren Erweiterung *SitRS XT* [FHWM16] stellt ein Beispiel für eine solche zentrale Verarbeitungskomponente für das Internet der Dinge dar.

Im Internet der Dinge haben demnach drei Instanzen Zugriff auf die darin enthaltenen Daten: (1) *Datenproduzenten* stellen Sensordaten zur Verfügung. (2) *Datenverarbeiter* leiten Ereignisse aus Sensordaten ab. (3) *Datenkonsumenten* reagieren auf Ereignisse. Insbesondere aufgrund der großen Anzahl an mannigfaltigen Sensoren, die in einer solchen Smart Environment miteinander verknüpft werden können, sind in einem solchen Anwendungsfall umfassende Maßnahmen hinsichtlich Datensicherheit und Datenschutz erforderlich [TUP16]. Infolge der verteilten Datennutzung, müssen IoT-Sicherheitslösungen auf allen drei Instanztypen arbeiten. Herkömmliche Sicherheitssysteme kommen mit einer solch heterogenen Umgebung jedoch nicht klar [Alq14]. Hat der Datenverarbeiter Zugriff auf alle Sensordaten, so kann das IoT zwar alle Ereignisse erkennen, jedoch verliert der Datenproduzent die Kontrolle über seine Daten. Stellt der Datenproduzent keine Daten zur Verfügung, sind diese zwar geschützt, jedoch hat der Datenkonsument in diesem Fall keinen Nutzen vom Internet der Dinge. Beides stellt keine zufriedenstellende Lösung dar. Eine sinnvolle Sicherheitslösung für das IoT muss daher folgende Aktivitäten unterstützen: (a) Daten von heterogenen Datenquellen sammeln, (b) Daten sicher speichern, (c) Daten gemäß Datenschutzrichtlinien verarbeiten, die vom Nutzer definiert wurden und (d) Daten Dritten zur Verfügung stellen ohne gegen die Datenschutzrichtlinien zu verstoßen [SBKN16].

Die *PMP* in Kombination mit dem *SDC+* unterstützt für einzelne Smart Devices diese vier Aktivitäten bereits vollständig (siehe Kapitel 4 und Kapitel 5).

Für das Internet der Dinge müsste das *Privacy Policy Model (PPM)* daher in einem verteilten System integriert und die *Privacy Rules* darin angewendet werden. In der eigenen Publikation [CSDM12] wird vorgestellt, wie solche Regeln in einem Datenstromsystem respektive einem CEP-System dazu verwendet werden können, um die Daten vor der Verarbeitung zu filtern oder die Verarbeitungsergebnisse nur aggregiert weiterzugeben. Aufgrund der feingranularen Beschaffenheit der *Privacy Rules* und den Konfigurationsmöglichkeiten, die die *Privacy Settings* bieten, können die Servicequalität und der Datenschutz austariert werden. Mit einer Cloud-basierten Variante des *SDC+*, wie sie im Rahmen der Diplomarbeit von Mayer [May15] umgesetzt wurde, können Daten auch in verteilten Systemen unter Berücksichtigung von Datensicherheitsaspekten gespeichert und verteilt werden.

Diese vier Anwendungsbeispiele für ein holistisches Sicherheitssystem vermitteln einen Eindruck davon, wie vielseitig die *PMP* und die ihr zugrundeliegenden Konzepte sind. In Kapitel 7 wird die *PMP* sowie der *SDC* respektive der *SDC+* unter mehreren Gesichtspunkten evaluiert, um zu ermitteln, ob die aufgestellten Schutzziele erfüllt werden (Abschnitt 7.1), wie die technischen Leistungsdaten sind (Abschnitt 7.2) und wie sich die *PMP* aus Nutzersicht darstellt (Abschnitt 7.3).

Kapitel 7

Evaluation der PMP und des SDCs

Obschon in Kapitel 4 und Kapitel 5 die Sicherheitsfeatures der *Privacy Management Platform (PMP)* und des *Secure Data Containers (SDCs)* dargelegt sowie deren Anwendbarkeit anhand von Anwendungsbeispielen in Kapitel 6 demonstriert wurden, stehen eine umfassende Evaluation, inwiefern diese Systeme die in Abschnitt 1.3 aufgestellten Schutzziele erfüllen, sowie Performanzmessungen noch aus. Im Folgenden wird daher in Abschnitt 7.1 zunächst analysiert, durch welche Features der *PMP*, des *SDCs* oder des *Secure Data Container Plus (SDC+)* welche Schutzziele erfüllt werden. Anhand eines Fallbeispiels aus dem Anwendungsbereich „mHealth“ wurden im Rahmen der vorliegenden Arbeit Performanzmessungen zu den drei Systemen durchgeführt. Die daraus hervorgegangenen Messergebnisse werden in Abschnitt 7.2 diskutiert. Wie die Untersuchungen von Felt et al. [FEF+12] und Felt et al. [FHE+12] zeigen, ist es für ein Datensicherheits- und Datenschutzsystem immanent wichtig, dass dieses vom Nutzer angenommen und verstanden wird. Daher wird in Abschnitt 7.3.1 besprochen, wie sich die *PMP* sowie der *SDC* respektive der *SDC+* aus Anwendersicht darstellt. Abschnitt 7.3.2 befasst sich

hingegen damit, wie die Akzeptanz seitens der Entwickler aussieht und welche Vorteile diese Systeme ihnen bieten. Abschließend werden in Abschnitt 7.4 die Erkenntnisse im Bezug auf den Stand der Technik zusammengefasst und die in der vorliegenden Arbeit herausgearbeiteten Konzepte von diesem abgegrenzt.

Dieses Kapitel basiert auf den eigenen Publikationen [SM13], [SM15] sowie [SM16a], wobei die Inhalte stark erweitert und auf die zwischenzeitlichen Veränderungen an den Plattformen angepasst wurden.

7.1 Erfüllung der aufgestellten Schutzziele

Ziel der in der vorliegenden Arbeit vorgestellten Konzepte ist es, ein holisitsches Sicherheitssystem für mobile Plattformen zu erstellen. Im ersten Schritt der Evaluation wird daher überprüft, ob die *PMP* und der *SDC* respektive der *SDC+* die in Abschnitt 1.3 aufgestellten Schutzziele an ein solches System erfüllen und inwiefern dies über die Sicherheitsfeatures von Android hinausgeht.

[**IS1**] Das Schutzziel der **Vertraulichkeit** wird gebrochen, wenn eine Instanz, d. h. ein Nutzer oder ein Prozess, Zugriff auf Daten erhält, für die er keine Autorisierung besitzt. Android stellt dies sicher, indem Apps logisch von den Android-Diensten (und damit auch von sämtlichen Zugriffen auf die Hardware) getrennt sind. In dem Anwendungsframework werden Application Programming Interfaces (APIs) bereitgestellt, die Apps über Inter-Process Communication (IPC)-Mechanismen Zugriff auf die Daten der Android-Dienste geben; für den Zugriff auf Daten von anderen Apps können *Content Provider* definiert werden. Da allerdings der Zugriff auf die APIs respektive die *Content Provider* – und dementsprechend auch der Zugriff auf die damit verbundenen Daten – nur sehr eingeschränkt reglementiert werden kann, gefährdet dies die Vertraulichkeit in hohem Maße [EOM09; OMEM09]. Durch den Einsatz der *PMP* erhält der Nutzer die volle Kontrolle über die Zugriffe auf *Resources*, die seine Apps ausführen. In der *Privacy Policy* legt er feingranular fest, welche App zu welchem Zweck Zugriff auf welche Daten erhält. Wird zur Speicherung von Daten der *SDC* verwendet, so werden diese automatisch verschlüsselt. Dies hat zur Folge, dass die Vertraulichkeit auch dann noch gewahrt wird, wenn ein Angreifer die Sicherheitsmechanismen des Betriebssystems aushebelt und illegal Zugriff auf den Datenspeicher erhält. Darüber hinaus lassen sich mit

dem *SDC* noch feingranularere Zugriffsregeln für jeden einzelnen Datensatz definieren, so dass der Einsatz der unsicheren *Content Provider* obsolet ist. Da Android als reines Einbenutzersystem konzipiert ist, findet, abgesehen von der Möglichkeit ein Smart Device zu sperren[1], keine Zugriffskontrolle auf Nutzerebene statt. Der *SDC+* löst dieses Problem, indem er Nutzerkonten und einen sicheren Authentifizierungsmechanismus einführt. Dadurch können sich mehrere Nutzer ein gemeinsames Smart Device teilen, ohne dass die Vertraulichkeit verletzt wird. Sogar ein sicherer Austausch von Daten unter den Nutzern ist damit möglich.

[**IS2**] Das Schutzziel der **Integrität**, bezogen auf den Ursprung sowie den Inhalt der Daten, wird in Android dadurch sichergestellt, dass sämtliche Sensoren nur über die APIs des Anwendungsframeworks angesprochen werden können. Das Anwendungsframework (respektive das darunterliegende Betriebssystem) muss sicherstellen, dass die Daten unverfälscht an die anfragenden Apps weitergegeben werden. Sofern das Betriebssystem nicht kompromittiert wird und Sensoren nicht mutwillig verfälschte Werte senden, ist die Integrität in Android daher für flüchtige Daten bereits gewährleistet. Dies gilt auch für die *PMP*, deren *Resources* die Unverfälschtheit der von ihr an Apps weitergegebenen Daten garantieren da alle *Resources* von vertrauenswürdigen Quellen stammen. Durch die Erweiterbarkeit der *PMP* lassen sich nachträglich weitere *Resources* nachrüsten, um so einen zusätzlichen Schutz für Datenquellen zu errichten, die bislang noch nicht reglementiert waren. Durch die Verschlüsselung der Daten gewährleisten der *SDC* und der *SDC+* zusätzlich, dass auch nichtflüchtige Daten in Datenbanken gegen Verfälschungen geschützt sind. Dies geht weit über den Schutz von Android hinaus, da hier die Daten der Datenbanken in Klartext abgespeichert sind.

[**IS3**] Das Schutzziel der **Verfügbarkeit** ist in Android dadurch sichergestellt, dass die Hersteller von Smart Devices das Betriebssystem und damit das Anwendungsframework beliebig anpassen können. Dadurch können diese die APIs für den Datenzugriff dahingehend ausweiten bzw. einschränken, dass darüber nur exakt die Daten erreichbar sind, die von der Hardware angeboten werden. Apps können explizit angeben, welche Schnittstellen sie voraussetzen, und in

[1] Untersuchungen von Harbach, De Luca und Egelman [HDE16] zeigen, dass diese Mechanismen allerdings unsicher sind und relativ leicht gebrochen werden können.

Google Play werden diese daraufhin nur für Smart Devices angeboten, die diese Schnittstellen unterstützen. Dadurch dass in Android keine Berechtigungen entzogen werden können, wird die Verfügbarkeit ebenfalls gewährleistet. Wird eine App installiert, besitzt diese alle zum Ausführen erforderlichen *Permissions* und kann daher zur Laufzeit auf alle benötigten Daten zugreifen. Die *PMP* (und damit auch der *SDC* sowie der *SDC+*) geht allerdings noch einen Schritt weiter. In jeder *Resource Group* können mehrere gleichartige *Resources* gebündelt werden. Beispielsweise kann die *Resource Group* zur Bestimmung des Standorts eine Global Positioning System (GPS)-basierte Implementierung, eine Global System for Mobile Communications (GSM)-basierte Implementierung sowie eine Implementierung auf Basis von Radio-Frequency Identification (RFID)-Markierungen verwenden. Dadurch können die Positionsdaten selbst dann noch an eine App weiter gegeben werden, wenn eines dieser Verfahren fehlschlägt. Durch die Spezifikation von separaten *Service Features* kann eine App zudem auf den Entzug von Berechtigungen reagieren und ihre Servicequalität entsprechend anpassen. Dadurch stehen ihr stets alle Daten zur Verfügung, die zu ihrer Ausführung nötig sind.

[**IS4**] Das Schutzziel der **Genauigkeit** ist unter Android voll erfüllt, da eine App die Daten stets in der bestmöglichen Qualität erhält; eine Einschränkung der Genauigkeit ist nicht vorgesehen. Da allerdings Apps zur Erfüllung ihrer Aufgaben die Daten nicht zwangsläufig mit einer maximalen Genauigkeit benötigen, widerspricht dies dem *Least-Privilege*-Prinzip. Bei der *PMP* (und damit auch im *SDC* sowie im *SDC+*) beschreibt eine App, mit welcher Genauigkeit sie Daten benötigt, um welche Services erfüllen zu können. Der Nutzer kann anschließend festlegen, wie genau eine *Resource* ihre Daten für diese App anbietet. Die *PMP* aktiviert anschließend die *Service Features* einer App, die mit der vom Nutzer festgelegten Genauigkeit erbracht werden können. Dadurch ist sichergestellt, dass die App alle Daten in der von ihr benötigten Genauigkeit erhält und gleichzeitig wird das *Least-Privilege*-Prinzip gewahrt.

[**IS5**] Das Schutzziel der **Nachvollziehbarkeit** ist in Android nur stark eingeschränkt erfüllt. Android persistiert lediglich, welche *Permissions* einer App gewährt wurden. Wozu eine App diese Berechtigungen nutzt, ist hingegen nicht dokumentiert. Die *PMP* hingegen protokolliert sämtliche Zugriffe einer App auf *Resources* (und damit auf Daten). Zusätzlich ist hinterlegt, wann einer App

gewisse Berechtigungen gewährt respektive entzogen wurden. Durch die Verschlüsselung im *SDC* respektive im *SDC+* ist darüber hinaus sichergestellt, dass keine illegalen Zugriffe auf die Daten erfolgen können – somit ist sichergestellt, dass alle Zugriffe nur über die *PMP* erfolgen können und daher protokolliert werden.

[**IS6**] Das Schutzziel der **Authentizität** ist in Android nicht erfüllt; Android gestattet die Installation von beliebigen Apps unbekannter Herkunft. Einzig die einer App zugewiesenen *Permissions* werden analysiert und es wird geprüft, ob diese zur Ausführung der App ausreichen. Da Apps allerdings *Permissions* untereinander austauschen können und nicht geprüft wird, von welchen Apps diese stammen, reicht ein solches Verfahren nicht aus. Die *PMP* (und damit auch der *SDC* sowie der *SDC+*) identifizieren und registrieren sämtliche Apps und *Resources*. Dadurch wird ebenfalls erfasst, von welchen *Resources* welche Apps Daten anfordern. *Privacy Rules* sind stets eindeutig an eine App gebunden, wodurch die Echtheit und Rechtmäßigkeit dieser gegeben ist. Der *SDC+* geht sogar noch einen Schritt weiter und identifiziert einzelne Nutzer, wodurch die Authentizität vollständig gewährleistet werden kann.

[**IS7**] Das Schutzziel der **Verbindlichkeit** ist in Android ebenfalls nicht erfüllt, da hier kein Protokoll über tatsächlich erfolgte Datenzugriffe erzeugt wird. Für die *PMP* und den *SDC* respektive den *SDC+* gelten die gleichen Argumente wie für das Schutzziel [IS5]. Da die *PMP* alle Zugriffe auf die *Resources* protokolliert, lässt sich jeder Zugriff einer App verbindlich belegen. Durch die Verschlüsselung der Daten ist garantiert, dass das Protokoll vollständig ist.

[**IS8**] Das Schutzziel der **Autorisierung** ist in Android nur unzureichend erfüllt. Die Autorisierung erfolgt ausschließlich über die einmalige Eingabe eines graphischen oder alphanumerischen Codes, um das Smart Device entsperren zu können. Dieses Vorgehen ist allerdings sehr unsicher [HDE16] und es kann keine Identifikation des Nutzers erfolgen. Die *PMP* identifiziert zumindest die Apps, die auf dem Smart Device ausgeführt werden, wodurch diese sich eindeutig gegenüber der *PMP* autorisieren können. Der *SDC+* führt neben Nutzerkonten einen sicheren Authentifizierungsmechanismus ein, wodurch sich auch Nutzer gegenüber dem System autorisieren können.

[**IS9**] Das Schutzziel der **Pseudonymität** ist in Android nicht erfüllt; wird eine App installiert, hat sie automatisch vollen und unverfälschten Zugriff

Schutzziel	Android	PMP	SDC	SDC+
[IS1] Vertraulichkeit	◔	◑	◕	●
[IS2] Integrität	◔	◑	●	●
[IS3] Verfügbarkeit	◑	●	●	●
[IS4] Genauigkeit	◕	●	●	●
[IS5] Nachvollziehbarkeit	◔	◑	●	●
[IS6] Authentizität	○	◑	◑	●
[IS7] Verbindlichkeit	○	◑	●	●
[IS8] Autorisierung	◔	◑	◑	●
[IS9] Pseudonymität	○	●	●	●

Tabelle 7.1: Erfüllungsgrad der Datensicherheitsziele (in Anlehnung an [SM16a])

auf alle Daten eines Nutzers. Dadurch werden der App viele Informationen zur Identität des Nutzers preisgegeben. Die *PMP* (und damit auch der *SDC* sowie der *SDC+*) gestattet es dem Nutzer hingegen, sämtliche Daten einer *Resource* randomisiert oder verfremdet an eine App weiterzuleiten, um so sogar vollständige **Anonymität** gewährleisten zu können.

Die Ergebnisse der Auswertung, inwiefern die Datensicherheitsziele von Android, von der *PMP*, vom *SDC* und vom *SDC+* erfüllt werden, sind in Tabelle 7.1 zusammengefasst. Bei der Bewertung kommt eine Maßskala mit fünf Abstufungen zum Einsatz. Der Füllgrad des dargestellten Kreises gibt an, wie gut ein Schutzziel von dem jeweiligen Ansatz erfüllt wird. Wie man sieht, erfüllt nur die Kombination aus *PMP* und *SDC+* alle Datensicherheitsziele vollständig.

[**DP1**] Das Schutzziel der **Einwilligung** ist in Android theoretisch erfüllt. Da ein Nutzer bei der Installation einer App allen angeforderten *Permissions* zustimmen muss, genehmigt er demnach automatisch jedweder Verarbeitung seiner Daten. Allerdings erteilt er die Berechtigungen unbewusst und gezwungenermaßen [FHE+12], wodurch diese Art der Einwilligung nicht im Sinne des Bundesdatenschutzgesetzes ist. Bei der *PMP* (und damit auch beim *SDC* sowie

beim *SDC+*) steht ihm zur Konfiguration der *Privacy Policy* eine nutzerfreundliche Graphical User Interface (GUI) zur Verfügung, über die er feingranular Berechtigungen an Apps verteilen und wieder entziehen kann. Bei dem *SDC+* kann der Nutzer sogar festlegen, welche anderen Nutzer Zugriff auf seine Daten erhalten und diese verarbeiten dürfen.

[**DP2**] Das Schutzziel der **Zweckbindung** ist in Android nicht erfüllt, da Apps Datenzugriffe gestattet werden, ohne dass diese einen Verwendungszweck angeben müssen. Darüber hinaus können sowohl Berechtigungen als auch Daten unkontrolliert zwischen Apps ausgetauscht werden [YZW+14]. Mit der *PMP* ist dies nicht möglich. Berechtigungen werden an eine App gebunden und diese muss explizit definieren, für welchen Zweck, d. h. für welches *Service Feature* sie bestimmte Daten benötigt. Der Nutzer kann einzelne *Service Features* deaktivieren, wodurch die hierfür benötigten Daten der App nicht länger zur Verfügung stehen. Mit dem *SDC* können Zugriffsrechte sogar auf Tupelebene vergeben werden, und mit dem *SDC+* kann eine Berechtigung sogar zusätzlich an einen bestimmten Nutzer gebunden werden.

[**DP3**] Das Schutzziel der **Erforderlichkeit** kann unter Android nicht gewährleistet werden. Eine App kann so viele *Permissions* anfordern, wie sie möchte, ohne dass nachvollziehbar ist, ob dies zur Erfüllung ihrer Funktionalität nötig ist. Auch kann der Nutzer hierauf keinen Einfluss nehmen. In der *PMP* (und damit auch im *SDC* sowie im *SDC+*) spezifiziert eine App über die *Service Features* sehr genau, welche Daten nötig sind, um welche Servicequalität zu erbringen. Der Nutzer kann den Leistungsumfang einer App beliebig reduzieren, wodurch auch die Menge ihrer Berechtigungen verringert wird. Dadurch kann die Menge der privaten Daten, auf die eine App Zugriff hat, auf ein Mindestmaß reduziert und die Erforderlichkeit kontrolliert werden.

[**DP4**] Das Schutzziel der **Transparenz** ist in Android nur unzureichend erfüllt. Ein Nutzer kann nicht nachvollziehen, auf welche seiner Daten eine App Zugriff erlangt und warum dies geschieht. Alles was ihm bei der Installation angezeigt wird, ist eine aggregierte und wenig aussagekräftige Sicht auf alle Berechtigungen, die eine App anfordert. Bei der *PMP* (und damit auch im *SDC* sowie im *SDC+*) erhält der Nutzer hingegen zur Laufzeit feingranulare Informationen über die App, welche *Service Features* gerade aktiv sind und auf welche Daten diese App mit welcher Genauigkeit zugreifen kann. Auch kann der

Nutzer die *Privacy Policy* jederzeit ändern, um Einfluss auf die Berechtigungen der App zu nehmen.

[**DP5**] Das Schutzziel der **Datensparsamkeit** ist in Android nur stark eingeschränkt erfüllt. Per se kann eine App auf keine privaten Daten zugreifen, wenn sie die hierfür benötigte *Permissions* nicht besitzt. Allerdings stellen *Permissions* stets Obermengen von vielen Berechtigungen dar, d. h. mit einer einzigen *Permission* darf eine App auf eine Vielzahl an Daten zugreifen. Zusätzlich ist ein Großteil der verfügbaren Apps überprivilegiert, was durch den Nutzer nicht reguliert werden kann [FCH+11]. In der *PMP* hingegen können Berechtigungen feingranular und auch zeitlich begrenzt oder an einen Kontext gebunden vergeben werden. Außerdem kann die Genauigkeit, mit der die Daten an eine App weitergegeben werden, auf ein Mindestmaß reduziert werden. Mit dem *SDC* ist dies sogar auf Tupelebene möglich, und mit dem *SDC+* können Berechtigungen sogar an einen Nutzer gebunden werden.

[**DP6**] Das Schutzziel der **Kontrolle** ist in Android nicht erfüllt. Ein Nutzer muss de facto einer App sämtliche Berechtigungen erteilen, wenn er diese verwenden möchte. Die *PMP* (und damit auch der *SDC* sowie der *SDC+*) gestattet es dem Nutzer hingegen, zur Laufzeit die *Privacy Policy* zu ändern und somit Einfluss auf die Berechtigungen der Apps zu nehmen. Darüber hinaus kann der Nutzer die Genauigkeit der Daten anpassen und sogar Daten beliebig verfremden oder randomisieren.

Die Ergebnisse der Auswertung, inwiefern die Datenschutzziele von Android, von der *PMP*, vom *SDC* und vom *SDC+* erfüllt werden, sind in Tabelle 7.2 zusammengefasst. Bei der Bewertung kommt eine Maßskala mit fünf Abstufungen zum Einsatz. Der Füllgrad des dargestellten Kreises gibt an, wie gut ein Schutzziel von dem jeweiligen Ansatz erfüllt wird. Wie man sieht, werden auch die Datenschutzziele nur von einer Kombination aus *PMP* und *SDC+* vollständig erfüllt.

Diese Analyse zeigt, dass die *PMP* erweitert um den *SDC+* ein holisitsches Sicherheitssystem für mobile Plattformen darstellt und somit die Forschungsziele voll erfüllt wurden. In Abbildung 7.1 ist ein Überblick über die wichtigsten Komponenten, die im Rahmen der vorliegenden Arbeit entwickelt werden, gegeben. Darin ist skizziert, wie diese Komponenten zur Erfüllung der Schutz-

Schutzziel	**Android**	**PMP**	**SDC**	**SDC+**
[DP1] Einwilligung	◑	◕	◕	●
[DP2] Zweckbindung	○	◑	◕	●
[DP3] Erforderlichkeit	○	●	●	●
[DP4] Transparenz	◔	●	●	●
[DP5] Datensparsamkeit	◔	◑	◕	●
[DP6] Kontrolle	○	●	●	●

Tabelle 7.2: Erfüllungsgrad der Datenschutzziele

ziele beitragen. Tragen mehrere Komponenten zur Erfüllung eines Ziels bei, so ist stets nur die essenziellste Komponente diesem Ziel zugeordnet.

Nachdem die Wirksamkeit der *PMP* und ihrer Erweiterungen gezeigt wurde, wird im Folgenden in einem zweiten Schritt die Performanz des Systems evaluiert.

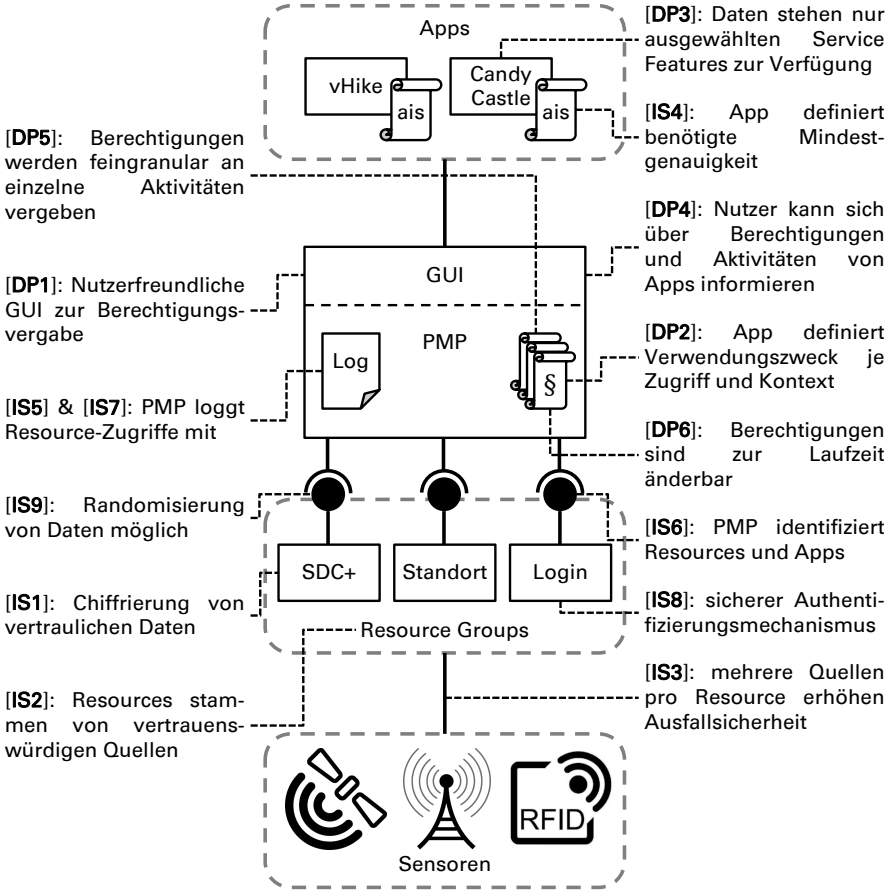


Abbildung 7.1: Erfüllung der Schutzziele durch die *PMP* und den *SDC+*

7.2 Technische Evaluation

Für die Durchführung der Performanzmessungen wurde ein realitätsnaher Anwendungsfall aus dem mHealth-Umfeld (siehe Abschnitt 6.2) herangezogen. Dabei geht es um die telemedizinische Überwachung von Patienten, die an Chronic Obstructive Pulmonary Disease (COPD), einer chronischen Lungenerkrankung leiden. Diese Patienten müssen regelmäßig mehrere Lungenfunktionen testen und Fragen zu ihrem aktuellen Befinden beantworten, um schwere

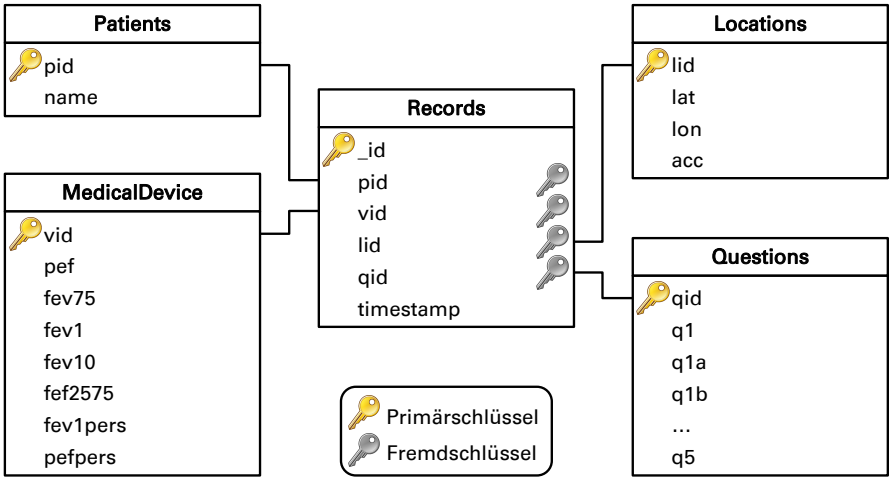


Abbildung 7.2: Datenschema eines Krankenakteneintrags der *ChronicOnline*-App (in Anlehnung an [SM16a])

Anfälle frühzeitig diagnostizieren und behandeln zu können. Im Rahmen des *ECHO*-Projekts[2] wurde zu diesem Zweck ein Cloud-Service entwickelt, der medizinische Daten verwalten und eine solche Diagnose durchführen kann. Zur Sammlung der Daten wurde mit *ChronicOnline*[3] zusätzlich eine Android-App entwickelt, mit der Patienten (*Patients*-Tabelle) ihr Befinden anhand von elf Fragen zu ihrer Gesundheit tagebuchartig festhalten können (*Questions*-Tabelle); die Antworten zu diesen Fragen werden mit den Lungenwerten verknüpft (*MedicalDevice*-Tabelle). Da Untersuchungen gezeigt haben, dass auch der Standort, an dem eine Messung durchgeführt wurde, für eine medizinische Analyse relevant ist [KM11], werden auch Standortdaten mit der elektronischen Krankenakte verknüpft (*Locations*-Tabelle). Das vollständige Datenschema für einen solchen Krankenakteneintrag (*Records*-Tabelle) ist in Abbildung 7.2 dargestellt. Diese Einträge werden zunächst in einer Datenbank der *ChronicOnline*-App zwischengespeichert und dann en gros an das Cloud-Back-End übermittelt.

[2] siehe http://chroniconline.eu/
[3] siehe https://play.google.com/store/apps/details?id=gr.openit.echo

Bei den Benchmark-Tests, mit denen die Performanzmessungen der *PMP*, des *SDCs* sowie des *SDC+* durchgeführt wurden, kam dieses Datenschema zum Einsatz. In dem Benchmark werden zwei unterschiedliche Szenarien simuliert. Zunächst wird die interne Datenbank einer App mit Krankenakteneinträgen befüllt und anschließend sollen diese Einträge von anderen Apps gelesen werden. Zur Speicherung kommen der *PMP-Datencontainer (PDC)*, der *SDC* sowie der *SDC+* zum Einsatz. Als Vergleichswert werden die Schreib- und Lesevorgänge auch mit normalen Android-Techniken, d. h. einem Zusammenspiel einer SQLite-Datenbank und *Content Providern*, ohne den Einsatz der *PMP* durchgeführt. Die Testsuite besteht daher für jedes der vier Untersuchungsobjekte aus zwei separaten Test-Apps, einem Schreib-Benchmark und einem Lese-Benchmark.

Beide Benchmarks wurden mit unterschiedlich großen Datensätzen durchgeführt. Der kleinste Datensatz besteht aus 500 verschiedenartigen Krankenakteneinträgen, die jeweils einem anderen Patienten zugeordnet sind. In fünf Schritten, in denen der Datensatz, d. h. die Anzahl der darin enthaltenen Krankenakteneinträgen, jeweils verdoppelt wird, wird der größte Datensatz mit 16.000 Einträgen erreicht. Diese Größen wurden angelehnt an den *McObject's-TestIndex*-Benchmark für Android Datenbanken[4] gewählt.

Im Rahmen der Testsuite erzeugt der Schreib-Benchmark zunächst n zufällige Krankenakteneinträge, d. h. es werden n zufällige Patienten, n zufällige Standorte, n zufällige Lungenwerte sowie n zufällige Antworten auf die elf Gesundheitsfragen erzeugt. Diese Daten werden anschließend jeweils zu einem Krankenakteneintrag verknüpft, der einen zufälligen Zeitstempel erhält (mit $n \in \{500, 1.000, 2.000, 4.000, 8.000, 16.000\}$). Diese Krankeneinträge werden danach in dem Datenspeicher, der evaluiert werden soll, gespeichert, d. h. in einer konventionellen *SQLite*-Datenbank (*Android*), in dem *PDC* (*PDC*), in dem *SDC* (*SDC*) oder in dem *SDC+* (*SDC+*). Sowohl im *SDC* als auch im *SDC+* wird der Advanced Encryption Standard (AES)-256-Algorithmus zur Verschlüsselung der Daten verwendet, da dieser als ausreichend sicher für ein solches mHealth-Szenario angesehen werden kann [KKR10]. Die Daten in der Android-*SQLite*-Datenbank und im *PDC* werden unverschlüsselt gespeichert.

[4] siehe http://www.mcobject.com/android

Nachdem der Schreib-Benchmark vollständig durchgeführt wurde, beginnt der Lese-Benchmark. Dieser greift auf alle zuvor geschriebenen Krankenakteneinträge in zufälliger Reihenfolge zu. Da es sich bei den beiden Benchmarks um zwei separate Apps handelt, muss der Schreib-Benchmark Mechanismen anbieten, mit denen er seine Daten mit dem Lese-Benchmark teilen kann. Im Fall von Android geschieht dies über *Content Provider*, während beim *PDC*, beim *SDC* und beim *SDC+* auf deren eigene Mechanismen zum Datenaustausch zurückgegriffen wird. Beide Benchmarks wurden jeweils zehnmal durchgeführt. Vor jeder Ausführung des Scheib-Benchmarks wurden dessen Datenbank sowie dessen Cache vollständig gelöscht, um Seiteneffekte ausschließen zu können[5].

Zur Durchführung der Messung kam ein Motorola Moto E Phone (zweite Generation) mit einer Qualcomm Snapdragon 410 CPU (1,2 GHz Quad Core), einem GB RAM und einer Akkukapazität von 2.390 mAh zum Einsatz. Diese Hardware stellte zum Zeitpunkt der Anfertigung der vorliegenden Arbeit ein durchschnittliches Mittelklasse-Smartphone dar. Aus diesem Grund können die Messergebnisse als repräsentativ angesehen werden. Das Betriebssystem des Smartphones wurde mit einem Vanilla-Android[6] der Version 5.1.1 und einem Kernel mit der Versionsnummer 3.10.87 ersetzt. Da Smartphone-Hersteller häufig Android für ihre Hardware anpassen und das Grundsystem um viele zusätzliche Dienste erweitern, können auf diese Weise unnötige Hintergrunddienste, die die Messergebnisse verfälschen könnten, weitestgehend ausgeschlossen werden. Dennoch bleiben die Ergebnisse realitätsgetreu, da keine Android-Dienste entfernt wurden.

Gemessen wurden in jedem Durchgang die Programmlaufzeit, die mittlere CPU-Auslastung, die Akkuentladung sowie der maximale Speicherverbrauch. Beim Speicherverbrauch wurden sowohl die *Proportional Set Size (PSS)* als auch die *Private Dirty Pages* berücksichtigt[7]. Als Messergebnis wurde jeweils der Median der zehn Durchläufe herangezogen, um eine Verfälschung der Messergebnisse durch Ausreißer ausschließen zu können. Jede Messung startet

[5] Befänden sich noch Daten im Cache, so können diese ggf. vom Lese-Benchmark verwendet werden, wodurch die Messergebnisse verfälscht werden würden.

[6] Unter Vanilla-Android oder auch Stock-Android versteht man die von Google entwickelte Grundversion von Android ohne spezielle Anpassungen oder zusätzliche Dienste.

[7] *Private Dirty Pages* berücksichtigen nur Speicherseiten, die ausschließlich von dem eigenen Prozess verwendet werden, während *PSS* auch Seiten berücksichtigt, die von mehreren Prozessen genutzt werden (siehe https://developer.android.com/studio/profile/investigate-ram.html).

Datensätze	Android	PDC	SDC	SDC+
Programmlaufzeit in Sekunden				
500 Tupel	34,5 s	35,8 s	38,6 s	38,4 s
16.000 Tupel	1.051,5 s	1.095,1 s	1.126,8 s	1.121,2 s
Akkuentladung in Mikroamperestunden				
500 Tupel	2,55 mAh	2,87 mAh	2,74 mAh	2,80 mAh
16.000 Tupel	78,19 mAh	86,62 mAh	81,89 mAh	80,84 mAh
Mittlere CPU-Auslastung in Prozent				
500 Tupel	32,9 %	31,0 %	29,8 %	30,6 %
16.000 Tupel	35,2 %	32,7 %	33,8 %	32,6 %
Maximaler Speicherverbrauch in Megabyte (PSS)				
500 Tupel	23,53 MB	20,41 MB	21,89 MB	21,65 MB
16.000 Tupel	26,11 MB	23,03 MB	24,73 MB	26,24 MB
Maximaler Speicherverbrauch in Megabyte (Private Dirty)				
500 Tupel	20,90 MB	16,05 MB	16,85 MB	16,81 MB
16.000 Tupel	23,36 MB	18,45 MB	19,53 MB	21,20 MB
Datenbankgröße in Kilobyte				
500 Tupel	120 kB	120 kB	136 kB	204 kB
16.000 Tupel	2.444 kB	2.444 kB	2.720 kB	4.800 kB

Tabelle 7.3: Benchmark-Ergebnisse – Schreib-Benchmark

unmittelbar bevor der erste Datensatz verarbeitet wird und endet, nachdem der letzte Datensatz vollständig verarbeitet wurde. Beim Schreib-Benchmark wird zusätzlich gemessen, wie groß die Datenbank wird; da der *SDC* sowie der *SDC+* zu jedem Datensatz zusätzliche Metadaten erzeugen, belegen diese mehr Speicher. Die Ergebnisse des Schreib-Benchmarks sind in Tabelle 7.3 aufgeführt und die des Lese-Benchmarks in Tabelle 7.4. Weitere Messergebnisse, wie die *Lines of Code* und die Größe der Benchmark-App sowie der benötigten *Resources* (im Fall des *PDCs*, *SDCs* sowie des *SDC+*) werden in Abschnitt 7.3 diskutiert (siehe Tabelle 7.5).

Datensätze	Android	PDC	SDC	SDC+
Programmlaufzeit in Sekunden				
500 Tupel	62,0 s	30,2 s	54,8 s	29,3 s
16.000 Tupel	2.011,6 s	972,7 s	1.367,1 s	936,3 s
Akkuentladung in Mikroamperestunden				
500 Tupel	4,59 mAh	2,41 mAh	4,30 mAh	2,06 mAh
16.000 Tupel	160,76 mAh	78,35 mAh	113,18 mAh	70,49 mAh
Mittlere CPU-Auslastung in Prozent				
500 Tupel	34,4 %	31,9 %	42,2 %	18,2 %
16.000 Tupel	34,9 %	29,9 %	42,6 %	17,9 %
Maximaler Speicherverbrauch in Megabyte (PSS)				
500 Tupel	19,67 MB	21,42 MB	22,26 MB	22,71 MB
16.000 Tupel	19,76 MB	24,14 MB	23,73 MB	23,23 MB
Maximaler Speicherverbrauch in Megabyte (Private Dirty)				
500 Tupel	16,84 MB	16,96 MB	17,52 MB	17,64 MB
16.000 Tupel	16,87 MB	19,57 MB	18,77 MB	18,03 MB

Tabelle 7.4: Benchmark-Ergebnisse – Lese-Benchmark

Die Auswertung der Messergebnisse führt zu mehreren Erkenntnissen: [**TA1**] Betrachtet man die **Programmlaufzeit** des Scheib-Benchmarks (siehe Abbildung 7.3), so spielt es praktisch keine Rolle, ob der *PDC*, der *SDC* oder der *SDC+* zum Einsatz kommt oder ob gänzlich auf die *PMP* verzichtet wird. Selbst der Einsatz der AES-256-Verschlüsselung im *SDC* sowie dem *SDC+* verursacht nur einen minimalen Overhead von etwa 7 %. Heruntergerechnet auf einen einzelnen Datensatz bedeutet dies, dass dessen Verarbeitung dadurch lediglich 5 Millisekunden länger dauert, obwohl sämtliche Datensicherheits- und Datenschutzfeatures (siehe Kapitel 4 respektive Kapitel 5) angewandt werden.

Betrachtet man hingegen die Programmlaufzeit des Lese-Benchmarks (siehe Abbildung 7.4), so spielen der *PDC*, *SDC* sowie *SDC+* ihre Stärken bezüglich eines sicheren Datenaustauschs aus. Hier zeigt sich deutlich, dass der Gebrauch

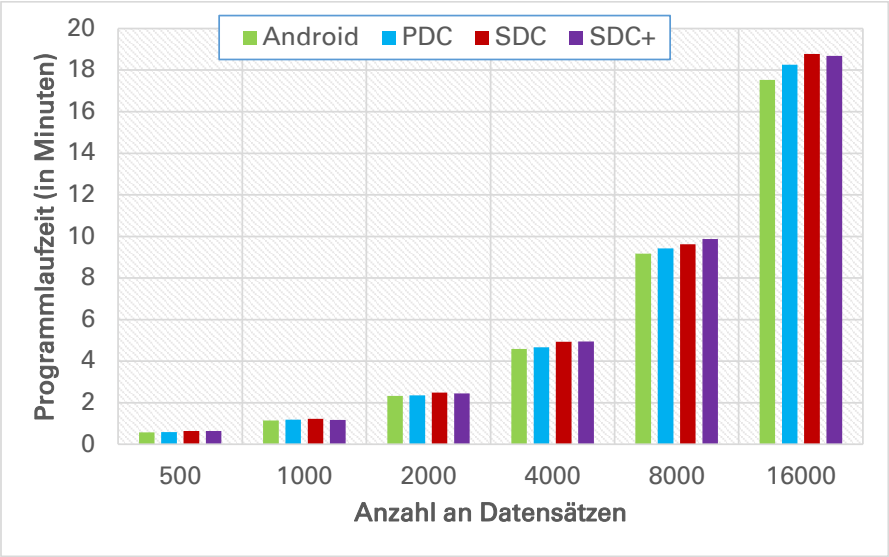


Abbildung 7.3: Grafische Übersicht der Messergebnisse: Programmlaufzeit – Schreib-Benchmark

eines *Content Providers* nicht nur sehr unsicher[8], sondern auch sehr rechenaufwendig ist. Die Client-App, d. h. in diesem Fall die Lese-Benchmark-App, muss zunächst den Besitzer der Daten, d. h. in diesem Fall die Schreib-Benchmark-App, über dessen *Content Provider* aufrufen. Dieser führt anschließend die Anfrage aus und leitet die Anfrageergebnisse an die Client-App zurück. Der *Content Provider* führt die hierfür nötige IPC automatisch durch. Die Messergebnisse zeigen hierbei eindeutig, dass der *PMP*-basierte Mechanismus zum Datenaustausch wesentlich performanter ist. Dabei ist der *PDC*, der ohne *Content Provider* auskommt, um mehr als 50 % schneller als der Android-basierte Ansatz. Selbst wenn der *SDC* für den Lese-Benchmark verwendet wird, ist dieser bei 16.000 Datensätzen trotz der von ihm benötigten Ver- und Entschlüsselungsoperationen um ca. 32 % schneller als der Android-basierte Ansatz. Da im *SDC+* die Daten von unterschiedlichen Patienten partitioniert werden können und somit die zu verwaltenden Datensätze pro Partition geringer sind,

[8] Beispielsweise hat der Besitzer der Daten, d. h. die App, die den *Content Provider* erzeugt, nahezu keine Reglementierungsmöglichkeiten, welche Apps Zugriff auf seine Daten nehmen [CFGW11].

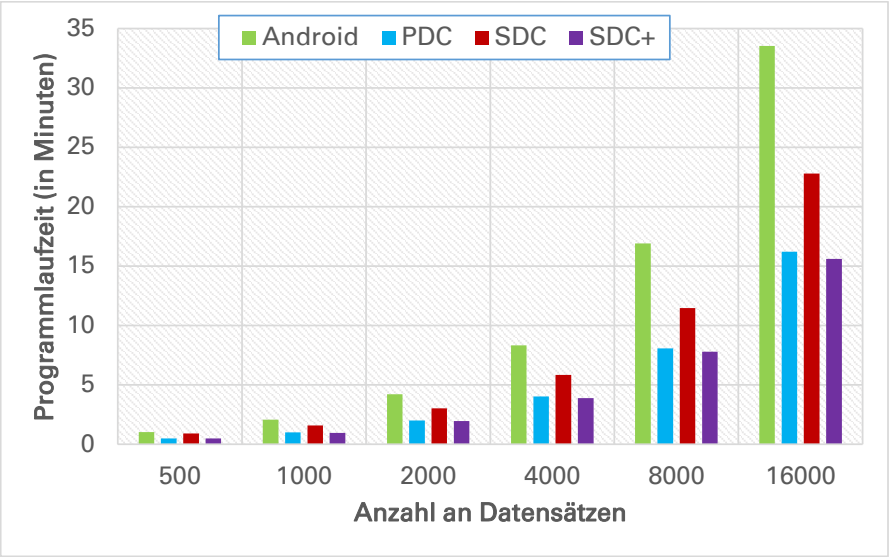


Abbildung 7.4: Grafische Übersicht der Messergebnisse: Programmlaufzeit – Lese-Benchmark

werden die Zugriffskosten weiter gesenkt. Dadurch übertrifft der *SDC+* sogar den *PDC* in Bezug auf die Programmlaufzeit. Insgesamt lässt sich feststellen, dass die Kosten für die Kryptographie immer vernachlässigbarer werden, je mehr Datensätze verarbeitet werden.

[**TA2**] Die mittlere **CPU-Auslastung** ist für den Schreib-Benchmark (siehe Abbildung 7.5) bei jedem der untersuchten Ansätze nahezu identisch. Für die *PMP*-basierten Ansätze wurden dabei alle beteiligten Prozesse, d. h. die Benchmark-App, die *PMP* selbst sowie alle benötigten *Resources*, berücksichtigt. Auch beim Lese-Benchmark (siehe Abbildung 7.6) sind der Android-basierte Ansatz (in diesem Fall werden die Kosten für den *Content Provider* berücksichtigt) und der *PDC* auf einem ähnlichen Level mit einem leichten Vorteil für den *PDC*. Da der *SDC* vor jedem Datenzugriff die komplette Datenbasis entschlüsseln und nach jedem Zugriff erneut verschlüsseln muss, wird hierbei die CPU stark beansprucht. Dies macht sich besonders beim Lese-Benchmark bemerkbar, da dieser viele einzelne Datenzugriffe ausführt. Trotz des dadurch verursachten Overheads ist die CPU-Auslastung des *SDCs* lediglich 22 % über der des

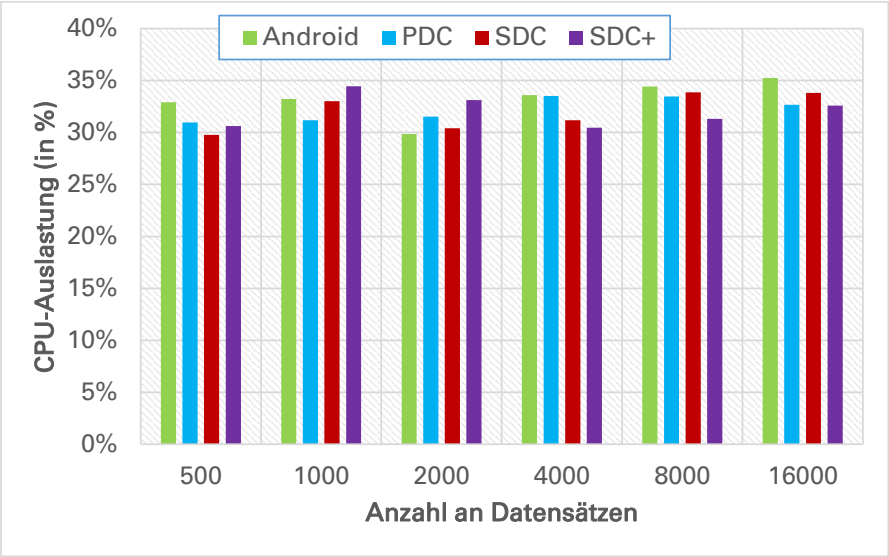


Abbildung 7.5: Grafische Übersicht der Messergebnisse: Mittlere CPU-Auslastung – Schreib-Benchmark

Android-basierten Ansatzes. Dank der Partitionierung des *SDC+* wird die zu verschlüsselnde Datenmenge pro Zugriff signifikant reduziert. Infolgedessen wird die CPU-Auslastung stark verringert, wodurch der *SDC+* den Android-basierten Ansatz aufgrund der darin nötigen rechenintensiven IPC-Aufrufe aussticht.

[**TA3**] Die **Akkuentladung** korreliert weitestgehend mit der Programmlaufzeit einer App und ihrer CPU-Auslastung. Da der Schreib-Benchmark (siehe Abbildung 7.7) für die vier untersuchten Ansätze in diesen beiden Kategorien ein ähnliches Verhalten zeigte, sind hier alle Ansätze auch bezüglich ihres Akkuverbrauchs gleichauf. Betrachtet man hingegen den Lese-Benchmark (siehe Abbildung 7.8), so wird der Android-basierte Ansatz von den drei *PMP*-basierten Ansätzen auch in dieser Kategorie ausgestochen. Besonders der *SDC+* muss in Bezug auf die Akkuentladung hervorgehoben werden, da dessen Akkuverbrauch weniger als halb so hoch wie der des Android-basierten Ansatzes ist. Ein Nutzer profitiert damit nicht nur von den erweiterten Datensicherheits- und Datenschutzfeatures des *SDC+*, sondern er spart auch noch Energie, d. h. er kann

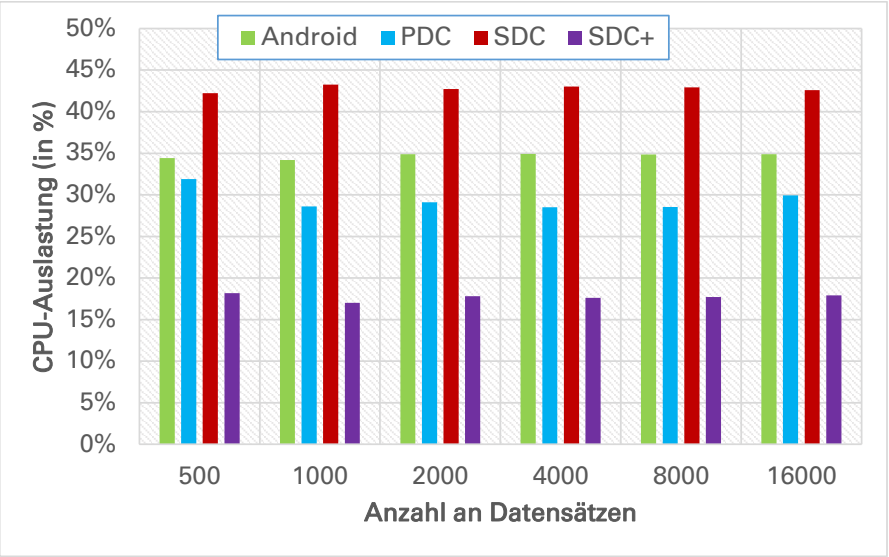


Abbildung 7.6: Grafische Übersicht der Messergebnisse: Mittlere CPU-Auslastung – Lese-Benchmark

seine App doppelt so lange nutzen, bevor er sein Smartphone an eine Stromquelle anschließen muss – aus Nutzersicht stellt dies einen unermesslichen Wert dar.

[**TA4**] Der maximale **Speicherverbrauch** berücksichtigt ausschließlich die Spitzenpegel, die während der Ausführung der Benchmark-Apps erreicht werden. Diese sind für den Nutzer besonders relevant, da sie beschreiben, wieviel freier Speicher zur Ausführung der App benötigt werden. Im Rahmen der Performanzmessungen werden sowohl die *PSS* als auch die *Private Dirty Pages* berücksichtigt. *PSS* rechnet sämtliche Speicherseiten mit ein, die von dem untersuchten Prozess belegt werden. Dies beinhaltet auch Speicherseiten, die sich dieser mit anderen Prozessen teilt[9]. *Private Dirty Pages* rechnen hingegen lediglich Speicherseiten ein, die ausschließlich von dem untersuchten Prozess belegt werden. Die Untersuchungen bezüglich dieser beiden Metriken zeigen allerdings sowohl für den Schreib-Benchmark (siehe Anhang Abbildung C.1a

[9] Geteilte Speicherseiten werden entsprechend der Belegung durch den untersuchten Prozess prozentual eingerechnet.

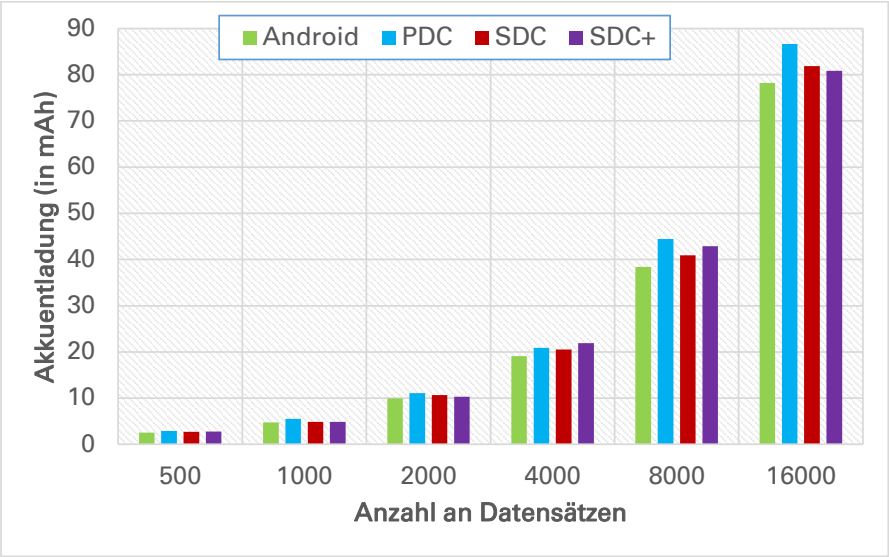


Abbildung 7.7: Grafische Übersicht der Messergebnisse: Akkuentladung – Schreib-Benchmark

und Abbildung C.2a) als auch für den Lese-Benchmark (siehe Anhang Abbildung C.1b und Abbildung C.2b) bei allen vier Ansätzen ein vergleichbares Verhalten. Daher wird im Folgenden nicht mehr explizit zwischen den beiden Metriken unterschieden.

Bei allen *PMP*-basierten Ansätzen (*PDC*, *SDC* sowie *SDC+*) werden die kumulierten Kosten berücksichtigt, d. h. es wird der Gesamtspeicherverbrauch der Benchmark-App, der *PMP* und aller beteiligter *Resources* berechnet. Beim Schreib-Benchmark liegen alle vier Ansätze erneut dicht beieinander. Im Lese-Benchmark wird durch den Gebrauch der *PMP* ein Overhead verursacht. Für die Entschlüsselung der Datenbank (*SDC* respektive *SDC+*) muss zusätzlicher Speicherplatz belegt werden. Besonders beim *SDC* macht sich dies bemerkbar, da hier die Datenbank nicht partitioniert und daher die zu entschlüsselnde Datei besonders groß ist. Bei diesem Overhead muss allerdings beachtet werden, dass eine einzige *PMP*-Instanz sehr viele Apps parallel verwalten und über die *Resources* mit Daten versorgen kann. In diesem Fall teilt sich der Speicherver-

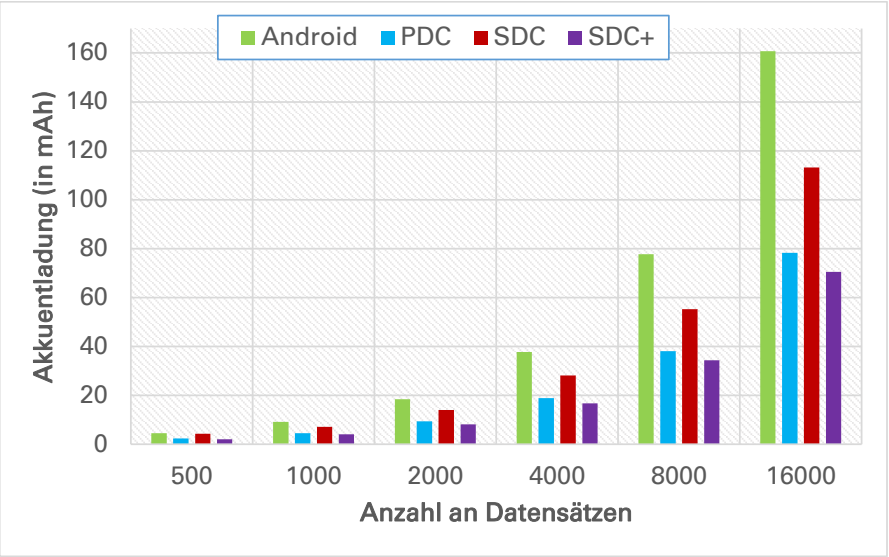


Abbildung 7.8: Grafische Übersicht der Messergebnisse: Akkuentladung – Lese-Benchmark

brauch der Verwaltungsaufgaben der *PMP*[10] gleichmäßig auf alle Apps, die die *PMP* nutzen, auf. Zieht man dies in Betracht, so würden die Analyse des Speicherverbrauchs für die *PMP*-basierten Ansätze wesentlich besser aussehen, wenn mehrere Benchmark-Apps parallel ausgeführt werden.

[**TA5**] Da der *SDC* respektive der *SDC+* zu jedem Datensatz zusätzliche Metadaten anlegt, die beschreiben, ob und mit welchen Apps dieser Datensatz geteilt wird (siehe Abbildung 5.3), steigt die **Datenbankgröße** bei diesen beiden Ansätzen wesentlich stärker an als bei dem Android-basierten Ansatz oder dem *PDC* (siehe Anhang Abbildung C.3). Der durch diese Verwaltungstabellen erzeugte Overhead macht beim *SDC* etwa 13 % für 500 Datenbankeinträge aus und verringert sich, je mehr Nutzdaten im *SDC* gespeichert werden. Beim *SDC+* ist dieser Overhead allerdings nicht zu vernachlässigen – die Verwaltungsdaten sind in etwa gleich groß wie die Nutzdaten. Dies ist der partitionierten Datenhaltung mit jeweils eigenen Indexstrukturen und unterschiedlichen Zugriffsrechten pro Partition geschuldet. In Anbetracht des Performanzgewinns

[10] Dies macht mehr als 50 % des gesamten Speicherverbrauchs der *PMP* aus.

und der erweiterten Datensicherheits- und Datenschutzfeatures, ist dies allerdings verschmerzbar, zumal die Speicherkapazitäten der Smart Devices kontinuierlich mit jedem neuen Modell zunehmen.

Die Performanzmessungen zeigen, dass der Einsatz des *SDCs* und insbesondere des *SDC+* nicht nur dann sinnvoll ist, wenn eine App vertrauliche Daten verarbeitet, sondern auch, wenn die Daten mit anderen Apps geteilt werden sollen. In diesem Fall wird ein gewaltiger Nutzeffekt in Bezug auf Programmlaufzeit, CPU-Auslastung, Akkuverbrauch und Speicherverbrauch generiert. Der durch die Chiffrierung entstehende Overhead kann durch die Partitionierung der Datenbank, wie es im *SDC+* möglich ist, mehr als aufgefangen werden. Darüber hinaus können im *SDC* und im *SDC+* auch andere, weniger rechenintensive Kryptographieverfahren verwendet werden (z. B. indem Schutzzonen eingeführt werden), wenn die darin gespeicherten Daten weniger vertraulich sind, um so die Performanz weiter zu steigern. Zusätzlich reduziert sich der Overhead der durch die Ver- und Entschlüsselung der Datenbank entsteht mit jeder Smart-Device-Generation von selbst, da der zusätzliche Rechenaufwand durch neue und schnellere Hardware aufgefangen wird. Dies zeigen frühere Evaluationsergebnisse auf einem Nexus One Dev Phone[11].

Nachdem die Performanz der *PMP* und ihrer Erweiterungen gezeigt wurde, wird im Folgenden in einem dritten Schritt eine nutzerorientierte Evaluation des Systems durchgeführt.

7.3 Bewertung aus Nutzersicht

Als Nutzer der *PMP* kommen zwei Gruppen in Betracht. Einerseits sind die Anwender, die Apps auf ihren Smart Devices verwenden, potentielle Nutzer der *PMP*. Eine Bewertung des Systems aus deren Sicht ist Gegenstand von Abschnitt 7.3.1. Andererseits sind die Entwickler von Apps ebenfalls Nutzer der *PMP*, da diese ihre Apps an die *PMP* anpassen müssen. In Abschnitt 7.3.2 wird daher das System unter deren Gesichtspunkten evaluiert.

Für beide Nutzergruppen gibt es ein paar relevante App-Metriken, die im Zuge der Benchmarks der technischen Evaluation (siehe Abschnitt 7.2) eben-

[11] Qualcomm QSD 8250 CPU, 1 GHz Single Core; 512 MB RAM; 1.400 mAh Akkukapazität; Vanilla-Android Version 2.3.6

Benchmark-App	Android	PDC	SDC	SDC+
Lines of Code				
Schreib-Benchmark	855 LOC	730 LOC	753 LOC	788 LOC
Lese-Benchmark	558 LOC	730 LOC	753 LOC	788 LOC
App-Größe in Kilobyte				
Schreib-Benchmark	448,5 kB	340,8 kB	338,6 kB	338,7 kB
Lese-Benchmark	445,8 kB	341,5 kB	340,3 kB	341,0 kB
Resource-Größe in Kilobyte				
Schreib-Benchmark	—	534,6 kB	536,2 kB	542,2 kB
Lese-Benchmark	—	534,6 kB	536,2 kB	542,2 kB
Gesamtgröße in Kilobyte				
Schreib-Benchmark	448,5 kB	875,4 kB	874,8 kB	880,9 kB
Lese-Benchmark	445,8 kB	876,1 kB	876,5 kB	883,2 kB

Tabelle 7.5: Benchmark-Ergebnisse – App-Metriken

falls gemessen wurden. Hierzu zählen die *Lines of Code*, die zum Erstellen einer App nötig sind, sowie die Größe der kompilierten App. Da im Fall der *PMP* zusätzlich eine oder mehrere *Resources* benötigt werden, wurde bei der Messung deren Größe bei der Gesamtgröße der App jeweils mitberücksichtigt. Die Ergebnisse sind in Tabelle 7.5 zusammengefasst sowie im Anhang in Abbildung C.4 grafisch veranschaulicht.

[**NA1**] Die ***Lines of Code*** stellen für einen Entwickler Arbeitsaufwand dar und für den Nutzer eine potentielle Fehlerquelle – je mehr Code geschrieben werden muss, desto mehr Fehler können gemacht werden. Daher ist diese Metrik für beide Gruppen von Interesse. Da der Besitzer einer Datenquelle unter Android *Content Provider* spezifizieren muss, fällt bei dem Android-basierten Schreib-Benchmark der Quellcode am längsten aus. Selbst für die *SDC+*-basierte App, die das zusätzlich kompliziertere Datenmodell definieren sowie die Partitionierung der Daten veranlassen muss, ist der Quellcode um beinahe 10 % kürzer als der der Android-basierten App. Für die Lese-Benchmarks ändert sich dieses Bild. Hier profitiert die Android-basierte App davon, dass

sie einfach die bestehenden *Content Provider* des Schreib-Benchmarks nutzen kann. Für die *PMP*-basierten Ansätze besteht zwischen dem Schreib- und Lese-Benchmark kein Unterschied, da darin lediglich die Schreiboperationen durch Leseoperationen ersetzt werden. Bezogen auf die reinen Zahlen ist der Android-basierte Lese-Benchmark um beinahe 30 % kürzer als die *SDC+*-basierte Variante. Allerdings muss in diesen Ansätzen nicht darauf geachtet werden, woher die Daten stammen, auf die zugegriffen werden soll, da alle Leseoperationen durchgeführt werden, als ob es sich um die Daten aus einer privaten Datenbank des Lese-Benchmarks handelt. Dadurch reduziert sich der Entwicklungsaufwand und die Fehleranfälligkeit insbesondere dann, wenn von vielen verschiedenen Apps Daten gelesen werden sollen.

[**NA2**] Die **App-Größe** ist besonders aus Nutzersicht entscheidend, da große Apps einerseits ein erhöhtes Datenvolumen beim Download verursachen und andererseits mehr Speicherplatz verbrauchen. Betrachtet man die reine App-Größe, so sind die Android-basierten Benchmark-Apps um etwa 25 % größer als die *PMP*-basierten Benchmark-Apps, obwohl alle *PMP*-basierten Apps zusätzlich ihr Application Information Set (AIS) definieren müssen. Das Verhältnis bei diesen sehr einfachen Apps zwischen Programmlogik und diesen Metadaten ist allerdings nicht sehr repräsentativ, da die Programmlogik hierbei ausgesprochen minimalistisch ist. In realitätsnahen Apps (siehe Kapitel 6) mit einer komplexen und umfangreichen Programmlogik, würden sich die Zahlen noch weiter zu Gunsten der *PMP*-basierten Apps verschieben. Nichtsdestoweniger benötigt jede *PMP*-basierte App noch zusätzliche *Resources*, die weiteren Speicherplatz belegen. Dadurch steigt die Gesamtgröße der *PMP*-basierten Benchmark-Apps auf nahezu die doppelte Größe der Android-basierten Benchmark-Apps an. Allerdings kann jede *Resource* von einer Vielzahl von Apps verwendet werden, wodurch der von ihr verursachte Overhead faktisch ignoriert werden kann.

7.3.1 Bewertung aus Anwendersicht

Nutzer von Smart Devices sind sich der wachsenden Bedrohung durch Schadsoftware bewusst. Jedoch sind sie der Meinung, dass viele ihrer Bedenken von den nativen Schutzsystemen der mobilen Plattformen, insbesondere dem von Android, nicht adressiert werden und dass diese Systeme ihnen nicht ausreichend Kontrolle über ihre privaten Daten geben [FHE+12; FEW12]. Aus

Anwendersicht erfüllt die *PMP* erweitert um den *SDC+* alle Erwartungen an ein holistisches Schutzsystem (siehe Abschnitt 7.1). Eine Nutzerstudie zur *PMP* wurde im Rahmen des Projekt-INFs von Fürst et al. [FHDZ13] durchgeführt. Nutzer konnten dabei die *PMP* ausprobieren. Das Konzept der *PMP* wurde dabei überwiegend positiv von den Anwendern bewertet. Einzig die große Zahl an Konfigurationsmöglichkeiten im Bezug auf die Berechtigungsvergabe wurde als kritisch angesehen, da die Erstellung einer *Privacy Policy* sehr aufwendig sein kann, wenn viele Apps von der *PMP* überwacht werden sollen. Durch den Einsatz von *Presets* kann dieser Aufwand jedoch erheblich reduziert werden was gerade unerfahrenen Anwendern sehr entgegen kommt.

Der *SDC* hingegen agiert aus Anwendersicht vollkommen transparent, d. h. die Datensicherheitsfeatures werden angewendet, ohne dass der Nutzer einen zusätzlichen Aufwand hat. Um die Nutzerkonten des *SDC+* verwenden zu können, ist eine einmalige Registrierung bei diesem sowie eine Authentifikation in jeder App, die diesen verwendet, nötig. Dies stellt allerdings keinen essenziellen Mehraufwand dar, da ein Login in jedem Fall nötig ist, wenn eine App mehrere Nutzer unterscheiden muss. Der sichere und universelle Login über die *PMP* stellt sogar einen Vorteil für den Nutzer dar, da er sich darüber gegenüber allen Apps authentifizieren kann und nicht für jede App separate Anmeldeinformation benötigt.

Insgesamt kann festgehalten werden, dass die *PMP* zusammen mit dem *SDC+* große Vorteile bezüglich Datenschutz und Datensicherheit für den Nutzer mit sich bringt, während sich der Overhead aus Anwendersicht stark in Grenzen hält.

7.3.2 Bewertung aus Entwicklersicht

Aus Entwicklersicht ist der reale Implementierungs-Overhead gering (siehe Tabelle 7.5), und die verfügbaren *Resources* nehmen dem Entwickler bereits viele Aufgaben ab (z. B. wird die aufwendige Erzeugung von Datenbanken und *Content Provider* durch den *SDC* obsolet). Auch stellt die notwendige Umstellung von Apps auf die *PMP* kein valides Gegenargument dar, da beispielsweise auch mit der Einführung der *Runtime-Permissions* in Android 6.0 Anpassungen an Apps nötig werden. Darüber hinaus existiert für App-Entwickler eine Werkzeugunterstützung (siehe Kapitel 8), die diesen bei der Arbeit mit der *PMP* hilft.

Dadurch wird nach einer kurzen Eingewöhnungszeit der Entwickler-Overhead weiter relativiert.

Da sehr viele Apps über personalisierte Werbung finanziert werden, benötigen diese unbegrenzten Zugriff auf private Nutzerdaten [ZGM12]. Durch die Verwendung eines holisitischen Sicherheitssystems wie die *PMP* kann der Nutzer diesen Zugriff einschränken oder vollständig verhindern, wodurch das Geschäftsmodell der Entwickler zerstört werden kann [LEPM12]. Allerdings stellt die *PMP* mit ihren *Service Features* auch eine Chance für die Entwickler dar. So können sie kostenlose Dienste anbieten, die keine Daten an Werbeanbieter weitergeben, und andere Premium-*Service-Features*, die zusätzliche Nutzerdaten anfordern und diese zu Werbezwecken verwenden. Ein Nutzer kann dadurch frei entscheiden, für welche Dienste er bereit ist, mit seinen Daten zu zahlen. Dieses Modell ist wesentlich transparenter und sollte insbesondere ehrlichen Entwicklern entgegenkommen.

7.4 Vergleich mit dem Stand der Technik

Da sich keine App von Haus aus vollständig an Privacy by Design (PbD) [Sha12] hält, ist sowohl ein Berechtigungssystem als auch ein Sicherheitssystem dringend erforderlich. Wie der Vergleich der mobilen Plattformen in Kapitel 2 zeigt, trifft dies insbesondere für Android zu. Eine Analyse des Stands der Technik zeigt, dass es sehr viele alternative Berechtigungssysteme für diese Plattform gibt (siehe Abschnitt 4.5). Allerdings ist die *PMP* das einzige Berechtigungssystem, das den Nutzer mit Feedback Informationen versorgt, Regeländerungen zur Laufzeit gestattet, Nutzerdaten verfremdet oder vollständig anonymisiert an Apps weiterleiten kann, im Falle eines Rechteentzugs zu keinem Absturz der App führt und kontextsensitive Berechtigungsregeln ermöglicht. Alle verwandten Arbeiten erfüllen jeweils nur eine Untermenge dieser Funktionen (siehe Tabelle 4.2). Auch im Bereich der Sicherheitssysteme existieren mannigfaltige Lösungsansätze (siehe Abschnitt 5.5). Während sich jedes um einen bestimmten Teilaspekt eines Sicherheitssystems kümmert (z. B. Berechtigungsvergabe, Datenverschlüsselung, sicheren Austausch von Daten, unwiederherstellbares Löschen von Daten, die Partitionierung von Daten in Sicherheitsstufen oder die

Einführung von separaten Nutzerkonten), ist der *SDC+* das einzige System, das alle diese Aspekte realisiert (siehe Tabelle 5.2).

Wie die Evaluation in Abschnitt 7.1 zeigt, können erst durch das Zusammenspiel von der *PMP* mit dem *SDC+* alle Schutzziele bezüglich Datenschutz und Datensicherheit vollständig erfüllt werden. Nachdem allerdings kein Ansatz die vollständige Funktionalität der *PMP* oder die des *SDC+* besitzt, gibt es auch keine verwandten Arbeiten, die dieses Synergiepotenzial besitzen. Damit gehen die in der vorliegenden Arbeit vorgestellten Konzepte weit über den heutigen Stand der Technik hinaus. Wie die Analyse der *PMP* und ihren Erweiterungen zeigt, erfüllt diese nicht nur sämtliche Schutzziele, sondern überzeugt auch hinsichtlich ihrer Performanz und bietet sowohl aus Anwendersicht als auch aus Entwicklersicht maßgebliche Vorteile.

Entwickler müssen sich bei der Implementierung ihrer Apps allerdings zunächst an den Umgang mit der *PMP* und an die Verwendung ihrer *Resources* gewöhnen. Daher werden in Kapitel 8 Werkzeuge für App-Entwickler vorgestellt, die die Arbeit mit der *PMP* erleichtern.

Kapitel 8

Werkzeugunterstützung für die PMP

Letzten Endes ist ein holistisches Schutzsystem wie die *Privacy Management Platform (PMP)*, das ein neues, erweitertes Berechtigungsmodell einführt und dadurch Anpassungen am Quellcode der Apps erforderlich macht, auf die Unterstützung der App-Entwickler angewiesen. Setzen diese das neue Modell nicht ein, so ist der Nutzen des Schutzsystems für den Anwender stark eingeschränkt. Mit dem *PMP-Gatekeeper* geht die *PMP* dieses Problem zwar teilweise an (siehe Abschnitt 4.4), der volle Funktionsumfang ist aber den Apps vorbehalten, die auf die *PMP* angepasst sind. Da die Evaluation in Kapitel 7 die Effizienz und Effektivität dieser Konzepte demonstriert, könnte ein Anbieter einer App-Plattform die *PMP* und ihre Erweiterungen, wie den *Secure Data Container Plus (SDC+)*, in seiner Plattform verwenden. Dadurch sind App-Entwickler gezwungen, das neue Berechtigungsmodell zu verwenden. Dies kann man beispielsweise an der Einführung der *Runtime Permissions* in Android 6.0 sehen (siehe Teegarden [Tee15]).

Zur Unterstützung der App-Entwickler wurden im Rahmen der vorliegenden Arbeit Werkzeuge und Konzepte entwickelt, die die Umstellung von bestehen-

den Apps auf die *PMP* sowie die Implementierung neuer Apps für die *PMP* erleichtern. Zuallererst ist hier das Konzept eines App-Konverters zu nennen, mit dem ein App-Entwickler seine bestehenden Apps automatisch auf die *PMP* anpassen kann, wie es in der eigenen Publikation [Sta15a] vorgestellt wurde. Dieses Konzept ist in Abschnitt 3.2 und Abschnitt 4.3.5 beschrieben. Bei der Implementierung neuer Apps hilft der *Editor for Information Sets (EIS)*, mit dem die Erstellung des *Application Information Sets (AISs)* und der *Resource Group Information Sets (RGISs)* erleichtert wird. Dieser Editor wird in Abschnitt 8.1 beschrieben. Abschnitt 8.2 beschäftigt sich mit einem Framework, das Entwicklern häufig benötigte Funktionen und Komponenten zur Verfügung stellt, wodurch die Implementierung von Apps erleichtert wird. Darüber hinaus wird ein Konzept vorgestellt, wie dieses Framework verwendet werden kann, um damit autoadaptive Apps zu entwickeln. Autoadaptive Apps passen sich selbstständig an Kontextänderungen an. Dadurch könnten Apps auf Änderungen der *Privacy Policy* reagieren, was durch die Aufteilung der Funktionalität in *Service Features* erleichtert wird.

Der *EIS* wurde im Rahmen des Studienprojekts von Vetter et al. [VJB+12] realisiert. *Gamework*, ein flexibles App-Framework, wurde in den eigenen Publikationen [Sta10a] sowie [Sta12] eingeführt und im Rahmen des Softwarepraktikums von Berg, Mack und Hilsbos [BMH10] prototypisch implementiert. Ein Konzept, wie damit autoadaptive Apps erstellt werden können, ist in der eigenen Publikation [Sta10b] beschrieben.

8.1 Eclipse-Erweiterung zur Erstellung der PMP-Metadaten

Aus Entwicklersicht besteht der größte Mehraufwand bei der Implementierung einer App, die die *PMP* verwendet, darin, dass zur Spezifikation der *Service Features* ein *AIS* erstellt werden muss. Hierfür muss der Entwickler eine Extensible Markup Language (XML)-Datei verfassen, die dem in Quelltext A.2 (siehe Anhang) angegebenen Schema entspricht. Darüber hinaus ist es erforderlich, dass die *PMP*-Registrierungsaktivität im Manifest der App definiert ist. Vergleichbare Schritte sind auch bei der Entwicklung von *Resources* nötig, nur dass hierbei das *RGIS* gemäß dem Schema in Quelltext A.1 (siehe Anhang) erstellt werden muss.

Die Anfertigung dieser Dateien ist nicht nur mühsam, sondern auch fehleranfällig. Beispielsweise kann das XML-Dokument nicht wohlgeformt oder valide sein und es können darin fehlerhafte Identifikatoren angegeben sein oder Einträge vergessen werden. Daher wurde mit dem *EIS* ein Werkzeug erstellt, das diesen Prozess vereinfacht. Auch lassen sich damit bestehende *AISs* und *RGISs* bearbeiten.

Anstelle dass Entwickler direkt mit dem XML-Quelltext arbeiten müssen, bietet der *EIS* mehrere Eingabemasken zur Konfiguration eines *AISs* und eines *RGISs*. Auf Knopfdruck wird eine neue, leere XML-Datei angelegt. In den Grundeinstellungen wählt der Entwickler ein *Eclipse*-Projekt aus, dem die Datei zugeordnet werden soll. Abhängig davon, ob es sich dabei um eine App oder eine *Resource* handelt und entsprechend ein *AIS* oder ein *RGIS* erstellt werden muss, unterscheiden sich die restlichen Felder der Grundeinstellungen nur marginal; die Grundeinstellungen entsprechen dem `appInformation`- respektive dem `resourceGroupInformation`-Element, weshalb die *RGIS*-Maske zusätzliche Felder hat (siehe Quelltext 4.4 bzw. Quelltext 4.1). Zu jedem Feld kann der Entwickler weitere Sprachen aus einer Dropdown-Liste auswählen und entsprechende Übersetzungen der Werte angeben. Bei der Eingabe der Daten wird geprüft, ob diese in dem erwarteten Wertebereich liegen und ggf. eine Fehlermeldung ausgegeben.

Für die Definition der *Service Features* (*AIS*) und der *Privacy Settings* (*RGIS*) gibt es jeweils eine separate Maske. Bei der *AIS*-Eingabemaske werden im linken Bereich alle aktuell definierten *Service Features* angezeigt. Über zwei Schaltflächen können weitere *Service Features* hinzugefügt oder bestehende gelöscht werden. Wird ein *Service Feature* ausgewählt, werden im rechten Bereich alle Details zu diesem angezeigt und können direkt bearbeitet werden. Neben einem Namen und einer Beschreibung – beides lässt sich in mehreren Sprachen angeben – kann ein Entwickler ein *Service Feature* um *Resources* ergänzen, die zu deren Ausführung nötig sind. Hierfür fragt der *EIS* beim Server der *PMP* an, welche *Resources* aktuell existieren. Wurde eine *Resource* ausgewählt, so übernimmt der *EIS* die Identifikations- und Paketinformationen automatisch in das *AIS* und ermittelt, welche *Privacy Settings* die *Resource* anbietet. Der Entwickler kann anschließend jedem *Privacy Setting* eine benötigte Konfiguration zuordnen. Fehlerhafte oder fehlende Daten werden farblich hervorgehoben.

Wurden alle Fehler behoben und alle obligatorischen Felder ausgefüllt, erzeugt der *EIS* auf Knopfdruck die XML-Dateien und fügt diese der App respektive der *Resource* zu. Dabei können auch vollautomatisch die Einträge in dem Manifest nachgezogen werden.

Die Eingabemaske für die *Privacy Settings* des *RGISs* ist ebenfalls zweigeteilt. Im linken Bereich findet sich eine Übersicht aller aktuell definierten *Privacy Settings*, die per Knopfdruck erweitert werden kann oder aus der bestehende Einträge entfernt werden können. Wurde ein Eintrag ausgewählt, so zeigt der *EIS* im rechten Bereich Details dazu an. Der Entwickler kann diese beliebig bearbeiten. Genau wie bei der Maske der *Service Features* findet hier automatisch eine Verifikation der Daten statt und fehlerhafte Werte werden markiert. Die *AIS*-Eingabemaske ist in Abbildung D.1 (siehe Anhang) dargestellt und Abbildung D.2 (siehe Anhang) zeigt die *RGIS*-Eingabemaske.

Android Apps können mit dem *Android Developer Tools (ADT)*-Plugin[1] für *Eclipse*[2] erstellt werden. Da sich App-Entwickler mit dieser Entwicklungsumgebung auskennen, wurde der *EIS* als *Eclipse*-Erweiterung implementiert. Die *EIS*-Oberfläche fügt sich nahtlos in *Eclipse* ein und orientiert sich an der Oberflächengestaltung des *ADT*-Plugins. Obwohl das *ADT*-Plugin von Google nicht mehr unterstützt wird, steht es Entwicklern weiterhin zur Verfügung. Dennoch wird seitens Google ein Umstieg auf *Android Studio*[3] empfohlen [Eas15]. Da *Android Studio* allerdings auf *IntelliJ IDEA*[4] basiert, lassen sich dafür analog zu *Eclipse* ebenfalls Plugins erstellen, d. h. der *EIS* kann auch dafür umgesetzt werden.

8.2 Framework für autoadaptive Apps

In der eigenen Publikation [Sta10a] wird mit *Gamework* ein Framework vorgestellt, das die Entwicklung von mobilen Apps weiter vereinfacht. Hierzu werden zunächst Funktionen identifiziert, die häufig in Apps benötigt werden. Diese werden anschließend in generischen Bausteinen zusammengefasst. Diese Bausteine können bei der Erstellung neuer Apps eingebunden werden, wodurch die

[1] siehe https://developer.android.com/studio/tools/sdk/eclipse-adt.html
[2] siehe https://eclipse.org/home/index.php
[3] siehe https://developer.android.com/studio/index.html
[4] siehe https://www.jetbrains.com/idea/

Entwicklung erheblich erleichtert wird. Die Bausteine lassen sich jeweils einer von fünf Kategorien zuordnen. So gibt es (1) Bausteine, mit denen Einfluss auf die Programmlogik genommen werden kann, (2) Bausteine zur Persistierung der Anwendungsdaten, (3) Bausteine, mit denen auf Kontextdaten zugegriffen werden kann, (4) Bausteine, mit denen Kontextdaten analysiert werden können, um beispielsweise Situationen daraus abzuleiten, sowie (5) Bausteine zur Gestaltung der Oberflächen von Apps.

Während die erste prototypische Implementierung von *Gamework*, die im Rahmen eines Softwarepraktikums von Berg, Mack und Hilsbos [BMH10] durchgeführt wurde, in erster Linie auf Spiele ausgerichtet ist, wird in der eigenen Publikation [Sta12] gezeigt, dass sich die Konzepte von *Gamework* auch auf andere App-Domänen übertragen lassen. Für die *PMP* können die aus dieser Vorarbeit gewonnenen Erkenntnisse dahingehend genutzt werden, dass neue Bausteine zur Persistierung von Daten sowie zur Erfassung des Kontextes erstellt werden, die hierfür auf die entsprechenden *Resources* zurückgreifen.

In *Gamework* wird die Programmlogik mittels eines endlichen Automaten angegeben. Ein Zustand beschreibt die Aktivitäten einer App. Jeder Zustand kann dabei mehrere Folgezustände haben, die abhängig von dem aktuellen Kontext oder einer Nutzereingabe ausgewählt werden. Dies ähnelt dem Prinzip der *Service Features*, in denen ebenfalls die Aktivitäten einer App gekapselt sind und die abhängig von der *Privacy Policy* ausgeführt werden. Das Konzept des endlichen Automaten kann daher für die *PMP* dafür genutzt werden, damit Entwickler möglichst einfach *Service Features* innerhalb ihrer App definieren können.

In der eigenen Publikation [Sta10b] wird *Gamework* konzeptionell um eine Feedback-Schleife erweitert, die dauerhaft Anwendungs- und Kontextdaten analysiert und darauf basierend Anpassungen an der App vornimmt. Diese Architektur von *Gamework* ist in Abbildung 8.1 dargestellt. Ein Beispiel, wie eine solche Schleife arbeiten kann, geben Rauchwerger und Amato [RA06] mit SmartApps. Bei der Kompilierung wird eine Monitoring-Komponente in die App eingebaut, die die Performanz zur Laufzeit überwacht. Abhängig von den Leistungsdaten wird die App angepasst, z. B. indem rechenintensive Aktivitäten übersprungen werden. Überwacht man anstelle der Performanz hingegen wie die *PMP* die Berechtigungen der App, so können Codeabschnitte,

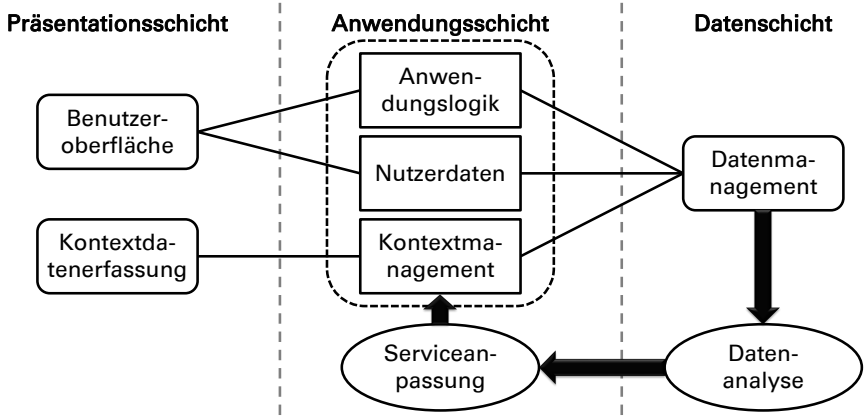


Abbildung 8.1: Architektur von *Gamework* (in Anlehnung an [Sta10b])

die mehr Berechtigungen benötigen, automatisch übersprungen werden. Auf diese Weise könnte eine App selbstständig *Service Features* kreieren, was den Entwickler entlasten würde. Auf diese Weise stellt *Gamework* einen ersten Schritt in Richtung autoadaptive Apps dar. Dies geht allerdings weit über den Fokus der vorliegenden Arbeit hinaus. Der interessierte Leser sei daher an dieser Stelle auf die Arbeit von Villegas Machado [Vil13] verwiesen.

Nachfolgend wird in Kapitel 9 eine Zusammenfassung der wesentlichen Erkenntnisse der vorliegenden Arbeit sowie ein Ausblick auf zukünftige Arbeiten in dem Forschungsumfeld dieser Arbeit gegeben.

Kapitel 9

Zusammenfassung und Ausblick

Zuletzt wird in diesem Kapitel ein Fazit aus der Arbeit gezogen und ein Ausblick auf zukünftige Arbeiten im Umfeld der *Privacy Management Platform (PMP)* gegeben. Die zukünftigen Arbeiten umfassen primär grundlegende Erweiterungen sowie Weiterentwicklungen, damit die in der vorliegenden Arbeit eingeführten Konzepte zum Management der Datensicherheit und des Datenschutzes auf andere Anwendungsdomänen, wie beispielsweise Complex Event Processing (CEP), angewandt werden können.

Zusammenfassung der Forschungsbeiträge

Smart Devices, wie beispielsweise Smartphones, gewinnen immer mehr an Bedeutung. Sie sind nicht nur ein dauerhafter Begleiter ihrer Nutzer, da sie mobil sind und eine lange Akkulaufzeit haben (*always on*), sondern sie sind auch fortwährend mit dem Internet verbunden und können dank dieses Umstands jederzeit Daten versenden und empfangen (*always connected*). Zusätzlich können diese Geräte über Sensoren den Kontext des Nutzers erfassen und Rückschlüsse

auf dessen gegenwärtige Situation ziehen. Aufgrund dieser Funktionalität, die weit über die herkömmlicher Desktop PCs hinausgeht, werden von Drittanbietern immer mehr Apps für unterschiedliche Anwendungsbereiche entwickelt. Neben reinen Unterhaltungs-Apps, wie beispielsweise Spiele, werden Smart Devices immer häufiger auch im medizinischen (*mHealth*) oder industriellen Umfeld (Industrie 4.0) eingesetzt.

Durch die Nutzung solcher Apps fallen allerdings sehr viele sensitive Daten an. Daher ist es wenig verwunderlich, dass Smart Devices immer mehr in den Fokus von Datendieben rücken. Dabei nimmt nicht nur die Anzahl an *Potentially Unwanted Applications (PUAs)* zu, wie beispielsweise Apps, die ihre Berechtigungen ausnutzen, um Nutzerdaten an Werbeanbieter weiterzuverkaufen, sondern auch die Anzahl an *Malware*, d. h. Apps, die bewusst Schwachstellen der Plattform ausnutzen, um illegal Zugriff auf sensible Daten zu erhalten. Begünstigt wird dies durch unzureichende Sicherheitsmaßnahmen seitens der mobilen Plattform und durch die Unerfahrenheit der Nutzer.

Das Ziel der vorliegenden Arbeit ist es daher, verbesserte Datensicherheits- und Datenschutzkonzepte für mobile Anwendungen zu erstellen. Hierfür werden die folgenden fünf Forschungsbeiträge erbracht:

Analyse bestehender Datensicherheits- und Datenschutzkonzepte (FB_1):

Kapitel 2 analysiert mit iOS, Windows Phone und Android die drei großen mobilen Plattformen, die zum Zeitpunkt der Anfertigung der vorliegenden Arbeit de facto den Markt unter sich aufteilen. Diese Analyse zeigt, dass alle Plattformanbieter zwar Datensicherheits- und Datenschutzmechanismen besitzen, diese aber nicht ausreichen, um alle relevanten Schutzziele (siehe Abschnitt 1.3) zu erfüllen. Insbesondere Android sticht hierbei heraus, da diese Plattform ein *User-Control-Sicherheitsmodell* verfolgt, d. h. sie überträgt dem Nutzer zwar die volle Verantwortung über seine Daten. Das darin verbaute Sicherheitssystem ist allerdings viel zu grobgranular und überfordert den Nutzer.

Daher beschreibt Kapitel 3 für Android, wie neue Datensicherheits- und Datenschutzkonzepte in die mobile Plattform eingebracht werden können. Dabei zeigt sich, dass entweder Anpassungen an der mobilen Plattform oder an den Apps nötig sind. Allerdings verstößt die Manipulation einer

App ohne das Einverständnis des Entwicklers gegen geltendes Recht. Daher scheidet dieses Verfahren für die Realisierung der im Rahmen dieser Arbeit entwickelten Konzepte aus.

Erstellung eines kontextsensitiven Berechtigungsmodells (FB_2):

Das Hauptproblem des Berechtigungssystems von Android liegt in dessen statischen und grobgranularen Berechtigungsmodells. Daher befasst sich der erste Teil von Kapitel 4 (genauer Abschnitt 4.1) mit der Einführung eines neuen kontextsensitiven und feingranularen Berechtigungsmodells. Das *Privacy Policy Model (PPM)* teilt Apps logisch in kleinere Funktionseinheiten, die *Service Features*, auf, die der Nutzer einzeln aktivieren respektive deaktivieren kann. Berechtigungen werden an *Service Features* und nicht komplett an Apps vergeben. Der Nutzer kann dadurch selbstständig festlegen, welche Funktionen er benötigt und welche Daten er dafür bereit ist mit der App zu teilen. Für die Zugriffe auf Datenquellen, wie Sensoren oder Datenspeicher, führt das *PPM Resources* ein. Der Nutzer kann über *Privacy Settings* die Zugriffe auf diese *Resources* regulieren und beispielsweise die Genauigkeit, mit der einer App Daten zur Verfügung gestellt werden, reduzieren. Ähnliche *Resources* werden in einer gemeinsamen *Resource Group* zusammengefasst. Dadurch kann eine App Daten von einer *Resource Group* anfordern, und die konkrete Implementierung, d. h. die *Resource*, wird erst zur Laufzeit bestimmt. In einer *Privacy Rule* wird angegeben, welche *Service Features* einer App auf welche Daten einer *Resource Group* zugreifen dürfen und welche *Privacy Settings* dabei angewandt werden sollen. Zusätzlich kann jede *Privacy Rule* mit einem Kontext versehen werden, in dem diese Regel gelten soll. Das *PPM* ermöglicht daher dem Nutzer eine feingranulare, flexible und kontextsensitive Berechtigungsvergabe.

Einbettung und Implementierung des Berechtigungsmodells in einem flexiblen Datenschutzsystem (FB_3):

Wie dieses Berechtigungsmodell in der *PMP* realisiert werden kann, zeigt der zweite Teil von Kapitel 4. In Abschnitt 4.2 wird diskutiert, welche Eigenschaften die *PMP* durch die Einbeziehung des *PPMs* besitzt. Die

PMP versorgt den Nutzer dauerhaft mit Feedback. Hierzu zählen u. a. Informationen zu den Auswirkungen von Veränderungen an den *Privacy Settings*. Fehlen einer App zur Laufzeit Berechtigungen, um die aktivierten *Service Features* auszuführen, benachrichtigt die *PMP* den Nutzer und erklärt ihm, welche zusätzlichen Daten benötigt werden, um das *Service Feature* dennoch ausführen zu können. Dadurch ist der Nutzer stets darüber im Bilde, welche Daten aktuell von welchen Apps verwendet werden können und zu welchem Zweck, d. h. für welches *Service Feature*, dieser Zugriff erforderlich ist. Mit diesen Informationen lässt sich die *Privacy Policy* exakt an die Nutzerwünsche anpassen. Die *Privacy Policy* kann jederzeit, auch zur Laufzeit einer App, angepasst werden. D. h. es können *Service Features* aktiviert oder deaktiviert sowie die *Privacy Settings* einer *Resource* verändert werden, wenn es erforderlich ist. Zu letzterem zählt auch die Genauigkeit, mit der die Daten von einer *Resource* angeboten werden. Diese kann soweit reduziert werden, bis die Daten vollständig anonymisiert oder randomisiert sind. Da *Service Features* stets optional sind und auf Wunsch deaktiviert, d. h. übersprungen werden können, führt ein Berechtigungsentzug nicht zu einem Absturz einer App; es werden nur die betroffenen *Service Features* ausgelassen. Jede *Privacy Rule* kann mit einem Kontext versehen werden, unter dem sie gültig ist. Ausgewählte *Privacy Rules* können in *Presets* ausgelagert werden, um diese auf anderen Geräten anwenden zu können. Weiterhin können so offizielle Stellen, wie das Bundesamt für Sicherheit in der Informationstechnik, empfohlene Konfigurationen für unerfahrene Nutzer anbieten. Da die *Resources* kein fester Bestandteil der *PMP* sind, können diese jederzeit installiert werden. Durch diese Erweiterbarkeit kann sich die *PMP* auf neue Gegebenheiten anpassen.

Abschnitt 4.3 untersucht, wie die *PMP* implementiert werden kann. Da keine Implementierungsstrategie insbesondere im Umgang mit *System-Apps* und *Legacy-Apps* vollständig überzeugen kann, wird in Abschnitt 4.4 mit dem *PMP-Gatekeeper* eine Komponente eingeführt, die diesen Nachteil ausmerzt. Mit dem *PMP-Gatekeeper* sind Berechtigungsänderungen zur Laufzeit für jeden App-Typ möglich, und zusätzlich greifen die Schutzfeatures der *PMP* für alle PMP-*kompatiblen Apps*. Der Vergleich mit anderen

Datenschutzsystemen in Abschnitt 4.5 zeigt, dass die Summe dieser Features ein Alleinstellungsmerkmal der *PMP* ist.

Erweiterung des Datenschutzsystems zu einem holistischen Sicherheitssystem (FB_4):

Die *PMP* stellt demnach bereits ein umfassendes Datenschutzsystem dar und sie wendet damit die von *PUAs* ausgehende Gefährdung ab. Im Rahmen der vorliegenden Arbeit wird die Erweiterbarkeit der *PMP* ausgenutzt und es werden zusätzliche *Resources* eingeführt, die Datensicherheitsfunktionen bereitstellen. Infolgedessen wird das System auch gegen Angriffe von *Malware* immun. Durch die Integration eines Datensicherheitssystems wird die *PMP* zu einem holistischen Sicherheitssystem. Zu diesem Zweck führt Kapitel 5 den *Secure Data Container (SDC)* ein. Der *SDC* kann von jeder beliebigen App genutzt werden. Damit dies möglich ist, kommt darin ein generisches und anpassbares Datenschema zum Einsatz. Sämtliche Daten werden vollständig verschlüsselt gespeichert und werden nur dann entschlüsselt, wenn eine berechtigte App darauf zugreift. Der Schlüssel wird nicht mit Apps geteilt, da diese potenzielle Gefahrenquellen darstellen und eine Schlüsselweitergabe ein Risiko bedeuten würde. Stattdessen verbleibt der Schlüssel im *SDC* und die *PMP* prüft, ob ein Datenzugriff genehmigt werden kann. Dadurch, dass alle Daten verschlüsselt vorliegen und es nur eine einzige Instanz des Schlüssels gibt, ist der *SDC* darüber hinaus in der Lage Daten sicher und nachhaltig zu löschen, indem die Schlüsselinstanz vernichtet wird; dies hat zur Folge, dass die Daten nicht mehr entschlüsselt und somit nicht mehr gelesen werden können. Dies ist hinsichtlich der Datensicherheit ein sehr wichtiges Feature, wenn das Smart Device kompromittiert wurde. Zusätzlich lassen sich die im *SDC* gespeicherten Daten in unterschiedliche Partitionen aufteilen und in jeder Partition eine andere Schutzzone oder ein anderes Datenschema anwenden. Ein weiteres wichtiges Feature des *SDCs* ist es, dass damit Daten sicher zwischen Apps ausgetauscht werden können. Auch wenn Android dies über *Content Provider* grundsätzlich ebenfalls ermöglicht, zeigen Untersuchungen, dass dies nicht nur umständlich, sondern auch unsicher ist. *Content Provider* gestatten es dem Datenbesitzer nicht, den

Zugriff auf seine Daten zu reglementieren. Im *SDC* ist dies vollständig anders. Nicht nur die Verwendung ist sehr einfach – der Zugriff auf fremde Daten unterscheidet sich nicht von dem auf die eigenen Daten – sondern der Datenbesitzer behält auch die volle Kontrolle über seine Daten.

In einem zweiten Schritt wird in Abschnitt 5.4 der *SDC* zum *Secure Data Container Plus (SDC+)* erweitert. Android ist als reines Einbenutzersystem konzipiert. Ein Smart Device kann allerdings auch von mehreren Nutzern verwendet werden. Daher führt der *SDC+* Nutzerkonten ein. Zugriffsrechte werden im *SDC+* nicht an Apps, sondern an eine Kombination aus Nutzer und App vergeben, d. h. mit dem *SDC+* ist es möglich, Daten zwischen Nutzern auszutauschen. Dies erfordert eine Nutzerauthentifikation, ohne dass bösartige Apps die Anmeldeinformationen abfangen können. Aus diesem Grund beinhaltet der *SDC+* ein sicheres Anmeldesystem als eigene *Resource*. Dieses System kann von anderen Apps eingebunden werden. Der Nutzer meldet sich dadurch direkt bei dem *SDC+* respektive der *PMP* an, ohne seine Anmeldeinformationen mit einer potenziell unsicheren App teilen zu müssen. Dieser sichere und universelle Login-Mechanismus kann auch zur Authentifikation innerhalb einer App verwendet werden. Die Analyse der verwandten Arbeiten in Abschnitt 5.5 zeigt, dass es aktuell kein umfassendes Datensicherheitssystem gibt, das vergleichbar zum *SDC+* ist. Verschiedene Arbeiten gehen jeweils einzelne Aspekte an, aber gerade die Kombination dieser Aspekte stellt ein Alleinstellungsmerkmal des *SDCs* sowie des *SDC+* dar.

Evaluation des holistischen Sicherheitssystems (FB_5):

Um den Nutzen und die Vielseitigkeit der in dieser Arbeit vorgeschlagenen Lösungen demonstrieren zu können, werden in Kapitel 6 Anwendungsbeispiele aus den Bereichen ortsbasierte Anwendungen, *mHealth*, Industrie 4.0 sowie Internet of Things (IoT) vorgestellt. In der anschließenden Evaluation in Kapitel 7 wird gezeigt, dass die Kombination aus *PMP* und *SDC+* nicht nur ein holistisches Sicherheitssystem ist, sondern auch alle relevanten Schutzziele erfüllt. Performanzmessungen zeigen darüber hinaus, dass das System dabei nicht nur sicher ist, sondern es auch einen gewaltigen Nutzeffekt in Bezug auf Programmlaufzeit,

CPU-Auslastung, Akkuverbrauch und Speicherverbrauch generiert. Aus Anwendersicht sind die Konzepte der vorliegenden Arbeit überzeugend. Auch für App-Entwickler bieten sie viele Vorteile; es entsteht jedoch ein geringfügiger Implementierungsmehraufwand. Daher wird in Kapitel 8 eine Werkzeugunterstützung für App-Entwickler vorgestellt.

Zusammenfassend ist festzuhalten, dass die in der vorliegenden Arbeit eingeführten Konzepte zum Management der Datensicherheit und des Datenschutzes alle in Abschnitt 1.3 aufgestellten Schutzziele erfüllen und die Forschungsbeiträge aus Abschnitt 1.4 dadurch erbracht wurden.

Ausblick

Da die im Rahmen der vorliegenden Arbeit entwickelten Prototypen in erster Linie als Machbarkeitsnachweis der vorgestellten Konzepte dienen, bedarf es weiterer Stabilitäts- und Kompatibilitätstests sowie einer umfassenden Usability-Studie, bevor die *PMP* im Produktivbetrieb eingesetzt werden kann. Im Folgenden werden noch zwei zukünftige Arbeiten im Forschungsumfeld der *PMP* beschrieben, die deren Funktionalität erweitern. Einerseits wird eine Vertrauensmetrik für die *PMP* eingeführt und andererseits werden die entwickelten Konzepte auf völlig neue Anwendungsdomänen übertragen.

Erweiterung der PMP um ein Reputationssystem

In der Bachelorarbeit von Aukschlat [Auk14] wird mit der *TriMetrik* eine Vertrauensmetrik für Apps eingeführt. Hierbei wird die Funktionalität einer App, die Art und Anzahl der von ihr angeforderten *Permissions* sowie die Art und Menge der von ihr erfassten sensiblen Daten bewertet und gewichtet miteinander verrechnet. Bei der Funktionalität einer App werden drei Aspekte berücksichtigt: (1) Wie haben andere Nutzer diese App bewertet? Hierbei werden neuere Bewertungen stärker gewichtet, da beispielsweise ein Update alte Probleme gelöst oder neue eingebracht haben könnte. (2) Wie häufig verwendet ein Nutzer diese App im Vergleich zu alternativen Apps mit der gleichen Funktionalität? (3) Wie viele Fehler wurden im Umgang mit dieser App in einem gegebenen Zeitintervall gemeldet? Diese Metrik wird im

Reputationssystem *TriTrust* prototypisch umgesetzt. Im Gegensatz zu vielen anderen Systemen ist *TriTrust* dabei vollständig kontextsensitiv, d. h. abhängig vom Nutzer, dessen Erfahrungen und der aktuellen Situation wird dynamisch entschieden, wie vertrauenswürdig eine App ist.

Eine zweckmäßige Umsetzung von *TriMetrik* setzt allerdings voraus, dass sowohl die mobile Plattform als auch der von ihr verwendete *App-Store* eng mit dem Reputationssystem zusammenarbeiten. Informationen über beispielsweise die Bewertungen anderer Nutzer oder Fehlermeldungen liegen nur im *App-Store* vor, während die Informationen über einen individuellen Nutzer nur auf dessen Smart Device zur Verfügung stehen. Zusätzlich wird eine App benötigt, die das Ergebnis der Bewertung anzeigt sowie dem Nutzer ermöglicht, die Gewichtung der Teilbewertungen vorzunehmen.

Hier kann die *PMP* eine zentrale Rolle einnehmen. Einerseits liegen ihr alle Informationen zu den Apps und deren Berechtigungen vor und andererseits kennt sie die Präferenzen des Nutzers in Bezug auf die Datennutzung von Apps. Darüber hinaus gibt die *PMP* dem Nutzer Feedback zu allen Berechtigungsfragen. In dieses Feedback könnten die *TriMetrik*-Bewertungen direkt eingebunden werden. Clientseitig müssten für die Einbindung von *TriTrust* daher primär Graphical User Interfaces (GUIs) angepasst und in der *TriMetrik* anstelle der *Permissions Privacy Settings* bewertet werden.

Das ebenfalls benötigte Back-End, d. h. ein *App-Store*, fehlt allerdings noch vollständig. In dem Projekt-INF von Loureiro, Prager und Zimmermann [LPZ14] wurde untersucht, wie ein *App-Store* aufgebaut werden kann. Aufbauend auf dieser Vorarbeit könnte im Rahmen von zukünftigen Arbeiten daher ein *App-Store* speziell für die *PMP* entwickelt werden, über den sowohl Apps als auch *Resources* bezogen werden können und der *TriTrust* mit allen benötigten Informationen versorgt. Einerseits wird die *PMP* dadurch um ein Reputationssystem erweitert und andererseits können Apps direkt in dem *App-Store* analog zu der Arbeit von Jarabek, Barrera und Aycock [JBA12] auf schadhaften Code überprüft werden; auf diese Weise würde die *PMP* sogar App-Prüfungen unterstützen (siehe Abschnitt 5.5).

Übertragung der Konzepte auf andere Anwendungsdomänen

Daten aus sozialen Netzwerken gewinnen immer mehr an Bedeutung. Die Nutzer dieser Netzwerke geben in ihrem Profil nicht nur sehr viele private Daten an und spezifizieren in welcher Beziehung sie zu den anderen Nutzern stehen, sondern sie erzeugen auch sehr viele User-Generated Contents (UGCs), beispielsweise in Form von Textnachrichten, Fotos oder Videos.

Spezielle Online-App-Plattformen stellen Smart-Device-Apps Dienste zur Verfügung, die die sozialen Daten aus dem Netzwerk mit Kontextdaten von dem Smart Device verknüpfen, um so die Servicequalität zu steigern. Litou, Boutsis und Kalogeraki [LBK14b] stellen einen solchen Dienst vor, der Nutzerbeziehungen, -daten und Kontextinformationen auswertet. Dieses Wissen kann dafür genutzt werden, um in Notfällen schnell potenzielle Betroffene identifizieren und informieren zu können [LBK14a]. Auch kann damit die Plausibilität der UGCs überprüft werden, um bewusste Fehlmeldungen gezielt löschen zu können, beispielsweise, wenn der Standort eines Nutzers nicht mit dem Ursprungsort des Inhalts übereinstimmt [LKBG16]. Allerdings kann dieses Wissen auch Verhaltensmuster des Nutzers aufdecken, die dieser nie preisgeben wollte. Beispielsweise kann man Rückschlüsse über soziale Beziehungen ziehen, wenn die Bewegungsprofile von zwei Nutzern viele Schnittpunkte aufzeigen, d. h. in einem solchen System werden neue Informationen über die Nutzer generiert. Die *PMP* kann in einem solchen Fall den Nutzern nicht helfen, da diese deren Daten nur schützen kann, so lange sie sich auf ihrem Smart Device befinden.

Im *PATRON*-Projekt[1] wird daher untersucht, wie private Informationen in einem Datenstromverarbeitungssystem geschützt werden können. Wie in der *PMP* muss es dabei einem Nutzer möglich sein, zu verstehen, welche Daten er einem Dienst zur Verfügung stellen muss, um welche Servicequalität zu erhalten [SDM+17]. Daher können die im Rahmen der vorliegenden Arbeit erarbeiteten Datensicherheits- und Datenschutzkonzepte als Basis herangezogen werden. In zukünftigen Arbeiten müssen diese auf derartige Online-Dienste angepasst werden. Ein reiner Zugriffskontrollmechanismus reicht hierbei nicht aus. Es muss zusätzlich sichergestellt werden, dass der Dienst auch keinen

[1] siehe http://patronresearch.de/

Zugriff auf andere Daten hat, aus denen er geschützte Informationen ableiten kann. Diese Regeln müssen nicht an einem zentralen Punkt, sondern in jedem Verarbeitungsknoten vorliegen, wie es in der eigenen Publikation [CSDM12] beschrieben ist.

Für die sichere Speicherung von Daten eines solchen Online-Diensts kann der *SDC+* als Basis für einen Cloud-basierten Datencontainer verwendet werden, wie es in der Diplomarbeit von Mayer [May15] prototypisch umgesetzt wurde. Wie es sich in der Evaluation des *SDC+* zeigt, belegen dessen Metadaten im Vergleich zu den eigentlichen Nutzdaten sehr viel Speicherplatz. Für die relativ kleinen Datenbestände auf Smart Devices stellt dies kein Problem dar. In einem Cloud-basierten Datencontainer mit sehr vielen Nutzern und demzufolge sehr vielen Nutzdaten, müssen die Metadaten auf eine andere Weise verwaltet werden. Zukünftige Arbeiten müssen daher untersuchen, wie alternative feingranulare Zugriffsmechanismen, z. B. *RestACL* von Hüffmeyer und Schreier [HS16], hierfür genutzt werden können.

Eigene Veröffentlichungen

[1] C. Stach und B. Mitschang. „CURATOR — A Secure Shared Object Store: Design, Implementation, and Evaluation of a Manageable, Secure, and Performant Data Exchange Mechanism for Smart Devices“. In: *Proceedings of the 33rd ACM/SIGAPP Symposium On Applied Computing: Database Theory, Technology, and Applications*. SAC ’18. ACM, Apr. 2018.

[2] C. Stach. „Big Brother is Smart Watching You: Privacy Concerns about Health and Fitness Applications“. In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. ICISSP ’18. SciTePress, Jan. 2018.

[3] C. Stach und B. Mitschang. „ACCESSORS: A Data-Centric Permission Model for the Internet of Things“. In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. ICISSP ’18. SciTePress, Jan. 2018.

[4] C. Stach, F. Steimle und B. Mitschang. „The Privacy Management Platform: An Enabler for Device Interoperability and Information Security in mHealth Applications“. In: *Proceedings of the 11th International Conference on Health Informatics*. HEALTHINF ’18. SciTePress, Jan. 2018.

[5] C. Stach, F. Steimle und A. C. Franco da Silva. „TIROL: The Extensible Interconnectivity Layer for mHealth Applications“. In: *Information and Software Technologies: 23nd International Conference, ICIST 2017, Druskininkai, Lithuania, October 12-14, 2017, Proceedings*. Herausgegeben von R. Damaševičius und V. Mikašytė. Band 756. Communications in Computer

and Information Science. Cham: Springer, Okt. 2017, Seiten 190–202. ISBN: 978-3-31967-642-5. DOI: 10.1007/978-3-319-67642-5_16.

[6] K. Mindermann, F. Riedel, A. Abdulkhaleq, C. Stach und S. Wagner. „Exploratory Study of the Privacy Extension for System Theoretic Process Analysis (STPA-Priv) to elicit Privacy Risks in eHealth“. In: *Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference Workshops, 4th International Workshop on Evolving Security & Privacy Requirements Engineering*. REW/ESPRE ’17. IEEE, Sep. 2017, Seiten 90–96. DOI: 10.1109/REW.2017.30.

[7] C. Stach, F. Dürr, K. Mindermann, S. M. Palanisamy, M. A. Tariq, B. Mitschang und S. Wagner. „PATRON — Datenschutz in Datenstromverarbeitungssystemen“. In: *Informatik 2017: Digitale Kulturen, Tagungsband der 47. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 25.9-29.9.2017, Chemnitz*. Band 275. LNI. GI, Sep. 2017, Seiten 1085–1096. ISBN: 978-3-88579-669-5. DOI: 10.18420/in2017_110.

[8] C. Giebler und C. Stach. „Datenschutzmechanismen für Gesundheitsspiele am Beispiel von Secure Candy Castle“. In: *Tagungsband der 15. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web*. BTW ’17. GI, März 2017, Seiten 311–320.

[9] C. Stach. „Secure Candy Castle — A Prototype for Privacy-Aware mHealth Apps“. In: *Proceedings of the 2016 IEEE 17th International Conference on Mobile Data Management*. MDM ’16. IEEE, Juni 2016, Seiten 361–364. DOI: 10.1109/MDM.2016.64.

[10] C. Stach und B. Mitschang. „The Secure Data Container: An Approach to Harmonize Data Sharing with Information Security“. In: *Proceedings of the 2016 IEEE 17th International Conference on Mobile Data Management*. MDM ’16. IEEE, Juni 2016, Seiten 292–297. DOI: 10.1109/MDM.2016.50.

[11] C. Gröger, C. Stach, B. Mitschang und E. Westkämper. „A mobile dashboard for analytics-based information provisioning on the shop floor“. In: *International Journal of Computer Integrated Manufacturing* (Mai 2016), Seiten 1–20. ISSN: 1362-3052. DOI: 10.1080/0951192X.2016.1187292.

[12] C. Stach. *Privacy_Management_Platform: Privacy Management Platform for Android* 6.0. GitHub Project. Okt. 2015. DOI: 10.5281/zenodo.32005. URL: https://github.com/stachch/Privacy_Management_Platform.

[13] C. Stach und B. Mitschang. „Der Secure Data Container (SDC) – Sicheres Datenmanagement für mobile Anwendungen“. In: *Datenbank-Spektrum* 15.2 (Juli 2015), Seiten 109–118. ISSN: 1618-2162. DOI: 10.1007/s13222-015-0189-y.

[14] C. Stach. „How to Deal with Third Party Apps in a Privacy System — The PMP Gatekeeper“. In: *Proceedings of the 2015 IEEE 16th International Conference on Mobile Data Management*. MDM ’15. IEEE, Juni 2015, Seiten 167–172. DOI: 10.1109/MDM.2015.17.

[15] C. Stach und B. Mitschang. „Design and Implementation of the Privacy Management Platform“. In: *Proceedings of the 2014 IEEE 15th International Conference on Mobile Data Management*. MDM ’14. IEEE, Juli 2014, Seiten 69–72. DOI: 10.1109/MDM.2014.14.

[16] C. Gröger und C. Stach. „The Mobile Manufacturing Dashboard“. In: *Proceedings of the 2014 IEEE International Conference on Pervasive Computing and Communications Workshops*. PerCom ’14. IEEE, März 2014, Seiten 138–140. DOI: 10.1109/PerComW.2014.6815180.

[17] C. Stach. „Wie funktioniert Datenschutz auf Mobilplattformen?“ In: *Informatik 2013: Informatik angepasst an Mensch, Organisation und Umwelt, Tagungsband der 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 16.9-20.9.2013, Koblenz*. Band 220. LNI. GI, Sep. 2013, Seiten 2072–2086. ISBN: 978-3-88579-614-5.

[18] C. Stach. „How to Assure Privacy on Android Phones and Devices?“ In: *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management*. MDM ’13. IEEE, Juni 2013, Seiten 350–352. DOI: 10.1109/MDM.2013.54.

[19] C. Stach und B. Mitschang. „Privacy Management for Mobile Platforms – A Review of Concepts and Approaches“. In: *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management*. MDM ’13. IEEE, Juni 2013, Seiten 305–313. DOI: 10.1109/MDM.2013.45.

[20] N. Cipriani, C. Stach, O. Dörler und B. Mitschang. „NexusDSS – A System for Security Compliant Processing of Data Streams“. In: *Proceedings of the First International Conference on Data Technologies and Applications*. DATA ’12. **DATA 2012 Best Paper Award**. SciTePress, Juli 2012, Seiten 175–185. DOI: 10.5220/0004051401750185.

[21] C. Stach und L. F. M. Schlindwein. „Candy Castle — A Prototype for Pervasive Health Games“. In: *Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops*. PerCom ’12. IEEE, März 2012, Seiten 501–503. DOI: 10.1109/PerComW.2012.6197547.

[22] C. Stach. „Gamework – A Framework Approach for Customizable Pervasive Applications“. In: *International Journal of Computer Information Systems and Industrial Management Applications* 4 (Jan. 2012), Seiten 66–75. ISSN: 2150-7988.

[23] C. Stach und A. Brodt. „vHike – A Dynamic Ride-sharing Service for Smartphones“. In: *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management*. MDM ’11. IEEE, Juni 2011, Seiten 333–336. DOI: 10.1109/MDM.2011.33.

[24] C. Stach. „Saving time, money and the environment – vHike a dynamic ride-sharing service for mobile devices“. In: *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops*. PerCom ’11. **IEEE PerCom 2011 Best WiP Poster Award**. IEEE, März 2011, Seiten 352–355. DOI: 10.1109/PERCOMW.2011.5766904.

[25] C. Stach. „Gamework – A Customizable Framework for Pervasive Games“. In: *Proceedings of the 2010 IADIS International Conference Game and Entertainment Technologies*. MCCSIS ’10. IADIS, Juli 2010, Seiten 45–52. ISBN: 978-972-8939-18-2.

[26] C. Stach. „Gamework – A customizable framework for pervasive games“. In: *Proceedings of the 7th International Conference on Pervasive Services*. ICPS ’10. ACM, Juli 2010, Seiten 168–173. ISBN: 978-1-4503-0249-4.

[27] A. Brodt und C. Stach. „Mobile ortsbasierte Browserspiele“. In: *Informatik 2009: Im Focus das Leben, Tagungsband der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 28.9.-2.10.2009, Lübeck*. Band 154. LNI. GI, Sep. 2009, Seiten 1902–1913. ISBN: 978-3-88579-248-2.

[28] C. Stach. „Mobile ortsbasierte Browserspiele“. Diplomarbeit. Universität Stuttgart : Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Germany, Jan. 2009.

Betreute Arbeiten

[1] F. Berg, F. Mack und P. Hilsbos. „Erschaffe Dein eigenes ‚mobile game'". Softwarepraktikum. Universität Stuttgart, Apr. 2010.

[2] B. Steeb und S. Kiesewetter. „vHike – a virtual hitchhiking system for mobile devices". Softwarepraktikum. Universität Stuttgart, Okt. 2010.

[3] H. Pfannkuch, J. Schlaak und S. Zitzelsberger. „Crowdsourcing: Einsatzgebiete, Bewertungssysteme und Motivationsmöglichkeiten". Fachstudie. Universität Stuttgart, Feb. 2011.

[4] F. Li und G. Sachs. „Could you take a photo?" Softwarepraktikum. Universität Stuttgart, März 2011.

[5] J. Dilli, T. Ferber und P. Jansa. „Privacy & Security – Wie machen das Apple, Google und Co.?" Fachstudie. Universität Stuttgart, Feb. 2012.

[6] M. Vetter, J. Jarosch, T. Berberich, H. D. Huynh, T. Kuhn, A. Makarov, A. B. Nguyen, F. Schäler, P. Strobel und A. Wassiljew. „Aufbau einer Privacy Management Platform für kontextsensitive mobile Apps". Studienprojekt. Universität Stuttgart, Juni 2012.

[7] F. Müller, Y. Noller und A. Schneider. „Marktanalyse zur App-Entwicklung". Fachstudie. Universität Stuttgart, Apr. 2013.

[8] P. Scholz. „Integration der PMP in das Android OS". Diplomarbeit. Universität Stuttgart, Apr. 2013.

[9] P. Metzger, E. Doust und J. Holzschuh. „Von Donuts, Pfefferkuchen und Sandwich-Eis". Fachstudie. Universität Stuttgart, Juni 2013.

[10] F. Schinkel, A. Schlag, G. Sachs, S. Haas und P. Schäfer. „Evaluation of Android Programming Enviroments". Projekt-INF. Universität Stuttgart, Juni 2013.

[11] A. Panos. „BYOD – Private Hardware in der Firma nutzen“. Diplomarbeit. Universität Stuttgart, Juli 2013.

[12] S. Fürst, M. Hoppe, F. K. Dang Huynh und J. Ziegler. „Evaluation of the Privacy Management Platform“. Projekt-INF. Universität Stuttgart, Aug. 2013.

[13] M. Steinert, P. Gerth und S. Mick. „LEGO Mindstorms – Spielend programmieren lernen“. Projekt-INF. Universität Stuttgart, Okt. 2013.

[14] M. d. O. Jullien. „Candy Castle : Um Jogo Sério para Pacientes com Diabetes“. Bachelorarbeit. Universidade Federal do Rio Grande do Sul, Dez. 2013.

[15] F. Schinkel. „Sichere mobile Gesundheitsspiele“. Bachelorarbeit. Universität Stuttgart, Dez. 2013.

[16] M. Braunbeck, P. Butz und C. Kleine. „Extension of a Privacy-Aware Ride-Sharing App“. Projekt-INF. Universität Stuttgart, Mai 2014.

[17] M. Loureiro, F. Prager und H. Zimmermann. „Design of an Online Marketplace for Mobile Applications“. Projekt-INF. Universität Stuttgart, Mai 2014.

[18] D. Salsa. „Der PMP Gatekeeper“. Bachelorarbeit. Universität Stuttgart, Aug. 2014.

[19] S. Speth, J. Schnapper, D. Michel, A. Frank und J. Thiel. „Design of a Data Container for mHealth Apps“. Projekt-INF. Universität Stuttgart, Nov. 2014.

[20] M. Aukschlat. „Vertrauen Jenseits Datenschutz und Datensicherheit“. Bachelorarbeit. Universität Stuttgart, Dez. 2014.

[21] M. Aukschlat, V. Jasny und S. Pirk. „Schutz privater Daten auf mobilen Geräten – geht das überhaupt?“ Fachstudie. Universität Stuttgart, Dez. 2014.

[22] K. Hatje. „Das Android Fragmentierungsproblem“. Diplomarbeit. Universität Stuttgart, Feb. 2015.

[23] A. Berchtold, T. Kieselmann und P. Wagner. „When Androids Control Robots“. Projekt-INF. Universität Stuttgart, Juni 2015.

[24] F. A. Mayer. „Ein sicherer Datencontainer für die Cloud“. Diplomarbeit. Universität Stuttgart, Dez. 2015.

[25] C. Giebler. „Privatheit im Gesundheitsspiel Candy Castle“. Bachelorarbeit. Universität Stuttgart, Jan. 2016.

[26] S. Pirk. „Privatheit im Internet der Dinge“. Bachelorarbeit. Universität Stuttgart, März 2016.

[27] D. Salsa und M. Gräber. „Native und hybride Entwicklung von kontextsensitiven Web-Apps“. Fachpraktikum. Universität Stuttgart, Apr. 2016.

[28] C. Braun, T. Boceck, A. Ayan, C. Bartsch, D. Brühl, D. Mair, A. Obralija, A. N. Tran, D. Tschechlov und S. Zeller. „JabRef 3 – modernes Bibliographiemanagement“. Studienprojekt. Universität Stuttgart, Sep. 2016.

[29] C. Giebler. „Erweiterungen für das eHealth-Spiel Secure CandyCastle“. Fachpraktikum. Universität Stuttgart, Okt. 2016.

[30] N. Krawietzek. „Eine sichere Schnittstelle zu Wearable Computers“. Bachelorarbeit. Universität Stuttgart, Jan. 2017.

[31] D. Salsa. „Ein sicherer Object Store für Android“. Masterarbeit. Universität Stuttgart, Juli 2017.

[32] C. Giebler. „LAKE — Eine flexible Datenstrom-Analyse-Architektur“. Masterarbeit. Universität Stuttgart, Okt. 2017.

Literaturverzeichnis

[AAS13] C. C. Aggarwal, N. Ashish und A. Sheth. „The Internet of Things: A Survey from the Data-Centric Perspective“. In: *Managing and Mining Sensor Data*. Herausgegeben von C. C. Aggarwal. Boston, MA, USA: Springer, 2013. Kapitel 12, Seiten 383–428. ISBN: 978-1-4614-6309-2. DOI: 10.1007/978-1-4614-6309-2_12 (Zitiert auf S. 233).

[ABK12] S. Avancha, A. Baxi und D. Kotz. „Privacy in Mobile Technology for Personal Healthcare“. In: *ACM Computing Surveys (CSUR)* 45.1 (Nov. 2012), 3:1–3:54. DOI: 10.1145/2379776.2379779 (Zitiert auf S. 226).

[ACD15] M. Ambrosin, M. Conti und T. Dargahi. „On the Feasibility of Attribute-Based Encryption on Smartphone Devices“. In: *Proceedings of the 2015 Workshop on IoT Challenges in Mobile and Industrial Systems*. IoT-Sys ’15. 2015, Seiten 49–54. DOI: 10.1145/2753476.2753482 (Zitiert auf S. 211).

[ACG15] W. Arthur, D. Challener und K. Goldman. „History of the TPM“. In: *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Berkeley, CA, USA: Apress, 2015, Seiten 1–5. ISBN: 978-1-4302-6584-9. DOI: 10.1007/978-1-4302-6584-9_1 (Zitiert auf S. 57).

[AEKR14] A. Arasu, K. Eguro, R. Kaushik und R. Ramamurthy. „Querying Encrypted Data“. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’14. 2014, Seiten 1259–1261. DOI: 10.1145/2588555.2588893 (Zitiert auf S. 214).

[AH13] Y. Agarwal und M. Hall. „ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing“. In: *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications,*

and Services. MobiSys '13. 2013, Seiten 97–110. DOI: 10.1145/2462456.2464460 (Zitiert auf S. 76).

[AHH+10] S. I. Ahamed, M. M. Haque, M. E. Hoque, F. Rahman und N. Talukder. „Design, Analysis, and Deployment of Omnipresent Formal Trust Model (FTM) with Trust Bootstrapping for Pervasive Environments". In: *Journal of Systems and Software* 83.2 (Feb. 2010), Seiten 253–270. DOI: 10.1016/j.jss.2009.09.040 (Zitiert auf S. 38).

[AJ10] M. Anvaari und S. Jansen. „Evaluating Architectural Openness in Mobile Software Platforms". In: *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. ECSA '10. 2010, Seiten 85–92. DOI: 10.1145/1842752.1842775 (Zitiert auf S. 58).

[AJP14] M. Aukschlat, V. Jasny und S. Pirk. „Schutz privater Daten auf mobilen Geräten – geht das überhaupt?" Fachstudie. Universität Stuttgart, Dez. 2014 (Zitiert auf S. 39).

[All15] G. Allen. „Android Security and Permissions". In: *Beginning Android*. Berkeley, CA, USA: Apress, 2015. Kapitel 20, Seiten 343–354. ISBN: 978-1-4302-4687-9. DOI: 10.1007/978-1-4302-4687-9_20 (Zitiert auf S. 62).

[Alq14] I. Alqassem. „Privacy and Security Requirements Framework for the Internet of Things (IoT)". In: *Companion Proceedings of the 36th International Conference on Software Engineering*. ICSE Companion '14. 2014, Seiten 739–741. DOI: 10.1145/2591062.2591201 (Zitiert auf S. 234).

[Ama13] R. Amadeo. *App Ops: Android 4.3's Hidden App Permission Manager, Control Permissions For Individual Apps! [Update]*. Pressemitteilung. Android Police, Juli 2013. URL: http://www.androidpolice.com/2013/07/25/app-ops-android-4-3s-hidden-app-permission-manager-control-permissions-for-individual-apps (Zitiert auf S. 119).

[AMN+13] M. S. Ahmad, N. E. Musa, R. Nadarajah, R. Hassan und N. E. Othman. „Comparison Between Android and iOS Operating System in terms of Security". In: *Proceedings of the 2013 8th International Conference on Information Technology in Asia*. CITA '13. 2013, Seiten 1–4. DOI: 10.1109/CITA.2013.6637558 (Zitiert auf S. 63).

[And16a] Android Developers. <*service*>. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/guide/topics/manifest/service-element.html#isolated (Zitiert auf S. 89).

[And16b] Android Developers. *Android Wear*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/wear/index.html (Zitiert auf S. 226).

[And16c] Android Developers. *Binder*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/reference/android/os/Binder.html (Zitiert auf S. 128).

[And16d] Android Developers. *Content Providers*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/guide/topics/providers/content-providers.html (Zitiert auf S. 186).

[And16e] Android Developers. *Creating a Content Provider*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/guide/topics/providers/content-provider-creating.html (Zitiert auf S. 186).

[And16f] Android Developers. *Cursor*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/reference/android/database/Cursor.html (Zitiert auf S. 186).

[And16g] Android Developers. *Processes and Threads*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/guide/components/processes-and-threads.html (Zitiert auf S. 130).

[And16h] Android Developers. *Requesting Permissions at Run Time*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/training/permissions/requesting.html (Zitiert auf S. 62).

[And16i] Android Developers. *Saving Data in SQL Databases*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://developer.android.com/training/basics/data-storage/databases.html (Zitiert auf S. 185).

[And16j] Android Open Source Project. *Android Interfaces and Architecture*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://source.android.com/devices/ (Zitiert auf S. 61).

[And16k] Android Open Source Project. *Full Disk Encryption*. Entwicklerunterlagen. Google Inc., Juli 2016. URL: https://source.android.com/security/encryption/ (Zitiert auf S. 60).

[AP12] AndroidPIT. *Fragwürdige Sicherheit durch SRT AppGuard?* Pressemitteilung. AndroidPIT International, Juli 2012. URL: https://www.androidpit.de/fragwuerdige-sicherheit-durch-srt-appguard (Zitiert auf S. 169).

[App16] Apple. *iOS Security*. Whitepaper. Apple Inc., Mai 2016. URL: https://www.apple.com/business/docs/iOS_Security_Guide.pdf (Zitiert auf S. 51, 53).

[APW17] S. Alpers, M. Pieper und M. Wagner. „Herausforderungen bei der Entwicklung von Anwendungen zum Selbstdatenschutz“. In: *Informatik 2017: Digitale Kulturen, Tagungsband der 47. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 25.9-29.9.2017, Chemnitz*. Band 275. LNI. 2017, Seiten 1061–1072. DOI: 10.18420/in2017_108 (Zitiert auf S. 85).

[AR98] P. E. Agre und M. Rotenberg, Herausgeber. *Technology and Privacy: The New Landscape*. 3rd printing. Cambridge, MA, USA: MIT Press, 1998. ISBN: 978-0-2625-1101-8 (Zitiert auf S. 36).

[ARAR16] A. Ali-Gombe, G. G. Richard III, I. Ahmed und V. Roussev. „Don’t Touch That Column: Portable, Fine-Grained Access Control for Android’s Native Content Providers“. In: *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. WiSec ’16. 2016, Seiten 79–90. DOI: 10.1145/2939918.2939927 (Zitiert auf S. 214).

[Auk14] M. Aukschlat. „Vertrauen Jenseits Datenschutz und Datensicherheit“. Bachelorarbeit. Universität Stuttgart, Dez. 2014 (Zitiert auf S. 39, 277).

[BAK+12] H. Banuri, M. Alam, S. Khan, J. Manzoor, B. Ali, Y. Khan, M. Yaseen, M. N. Tahir, T. Ali, Q. Alam und X. Zhang. „An Android Runtime Security Policy Enforcement Framework“. In: *Personal and Ubiquitous Computing* 16.6 (Aug. 2012), Seiten 631–641. DOI: 10.1007/s00779-011-0437-6 (Zitiert auf S. 79, 177).

[Bao16] A. Baokar. *A Contextually-Aware, Privacy-Preserving Android Permission Model*. Technischer Bericht UCB/EECS-2016-69. EECS Department, University of California at Berkeley, Mai 2016. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-69.html (Zitiert auf S. 62).

[Bau14] T. Bauernhansl. „Die Vierte Industrielle Revolution – Der Weg in ein wertschaffendes Produktionsparadigma“. In: *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung · Technologien · Migration*. Herausgegeben von T. Bauernhansl, M. ten Hompel und B. Vogel-Heuser. Wiesbaden: Springer, 2014, Seiten 5–35. ISBN: 978-3-6580-4682-8. DOI: 10.1007/978-3-658-04682-8_1 (Zitiert auf S. 229).

[BBGvS14] M. Backes, S. Bugiel, S. Gerling und P. von Styp-Rekowsky. „Android Security Framework: Extensible Multi-layered Access Control on Android“. In: *Proceedings of the 30th Annual Computer Security Applications Conference*. ACSAC ’14. 2014, Seiten 46–55. DOI: 10.1145/2664243.2664265 (Zitiert auf S. 80).

[BBH+15] M. Backes, S. Bugiel, C. Hammer, O. Schranz und P. von Styp-Rekowsky. „Boxify: Full-fledged App Sandboxing for Stock Android“. In: *Proceedings of the 24th USENIX Security Symposium*. USENIX Security ’15. 2015, Seiten 691–706 (Zitiert auf S. 88).

[BBK14] M. Braunbeck, P. Butz und C. Kleine. „Extension of a Privacy-Aware Ride-Sharing App“. Projekt-INF. Universität Stuttgart, Mai 2014 (Zitiert auf S. 224).

[BCLO09] A. Boldyreva, N. Chenette, Y. Lee und A. O’Neill. „Order-Preserving Symmetric Encryption“. In: *Advances in Cryptology - EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*. Herausgegeben von A. Joux. Berlin, Heidelberg: Springer, 2009, Seiten 224–241. ISBN: 978-3-6420-1001-9. DOI: 10.1007/978-3-642-01001-9_13 (Zitiert auf S. 213).

[BCMvO12] D. Barrera, J. Clark, D. McCarney und P. C. van Oorschot. „Understanding and Improving App Installation Security Mechanisms Through Empirical Analysis of Android“. In: *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM ’12. 2012, Seiten 81–92. DOI: 10.1145/2381934.2381949 (Zitiert auf S. 59).

[BD15] R. Bruns und J. Dunkel. *Complex Event Processing – Komplexe Analyse von massiven Datenströmen mit CEP*. Wiesbaden: Springer, 2015. ISBN: 978-3-6580-9899-5. DOI: 10.1007/978-3-658-09899-5 (Zitiert auf S. 234).

[BDDS15] M. Bucicoiu, L. Davi, R. Deaconescu und A.-R. Sadeghi. „XiOS: Extended Application Sandboxing on iOS“. In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS ’15. 2015, Seiten 43–54. DOI: 10.1145/2714576.2714629 (Zitiert auf S. 76).

[BDL14] Y. Bai, L. Dai und J. Li. „Issues and Challenges in Securing eHealth Systems“. In: *International Journal of E-Health and Medical Communications* 5.1 (Jan. 2014), Seiten 1–19. DOI: 10.4018/ijehmc.2014010101 (Zitiert auf S. 121, 226).

[Bea08] V. Beal. *The Difference Between a Cell Phone, Smartphone and PDA*. Pressemitteilung. webopedia, Mai 2008. URL: http://www.webopedia.com/DidYouKnow/Hardware_Software/smartphone_cellphone_pda.asp (Zitiert auf S. 29).

[BEvO11] D. Barrera, W. Enck und P. C. van Oorschot. *Seeding a Security-Enhancing Infrastructure for Multi-market Application Ecosystems*. Technischer Bericht TR-11-06. Carleton University, School of Computer Science, Apr. 2011. URL: https://www.scs.carleton.ca/node/93 (Zitiert auf S. 67).

[BFG10] M. Y. Becker, C. Fournet und A. D. Gordon. „SecPAL: Design and Semantics of a Decentralized Authorization Language“. In: *Journal of Computer Security – Digital Identity Management* 18.4 (Dez. 2010), Seiten 619–665. DOI: 10.3233/JCS-2009-0364 (Zitiert auf S. 87).

[BFTP15] L. Brutschy, P. Ferrara, O. Tripp und M. Pistoia. „ShamDroid: Gracefully Degrading Functionality in the Presence of Limited Resource Access“. In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. OOPSLA '15. 2015, Seiten 316–331. DOI: 10.1145/2814270.2814296 (Zitiert auf S. 227).

[BGF+10] G. Bai, L. Gu, T. Feng, Y. Guo und X. Chen. „Context-Aware Usage Control for Android“. In: *Security and Privacy in Communication Networks: 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings*. Herausgegeben von S. Jajodia und J. Zhou. Berlin, Heidelberg: Springer, 2010, Seiten 326–343. ISBN: 978-3-6421-6161-2. DOI: 10.1007/978-3-642-16161-2_19 (Zitiert auf S. 123).

[BGH+12] M. Backes, S. Gerling, C. Hammer, M. Maffei und P. von Styp-Rekowsky. *AppGuard – real-time policy enforcement for third-party applications*. Technischer Bericht A/02/2012. Universität des Saarlandes, 2012 (Zitiert auf S. 84, 168).

[BGH+13] M. Backes, S. Gerling, C. Hammer, M. Maffei und P. von Styp-Rekowsky. „AppGuard – Enforcing User Requirements on Android Apps“. In: *Tools and Algorithms for the Construction and Analysis of Systems: 19th International Conference, TACAS 2013, Held as Part of the European Joint*

Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. Herausgegeben von N. Piterman und S. A. Smolka. Berlin, Heidelberg: Springer, 2013, Seiten 543–548. ISBN: 978-3-6423-6742-7. DOI: 10.1007/978-3-642-36742-7_39 (Zitiert auf S. 168).

[BGH+14] M. Backes, S. Gerling, C. Hammer, M. Maffei und P. von Styp-Rekowsky. „AppGuard – Fine-Grained Policy Enforcement for Untrusted Android Applications". In: *Data Privacy Management and Autonomous Spontaneous Security: 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers*. Herausgegeben von J. Garcia-Alfaro, G. Lioudakis, N. Cuppens-Boulahia, S. Foley und M. W. Fitzgerald. Berlin, Heidelberg: Springer, 2014, Seiten 213–231. ISBN: 978-3-6425-4568-9. DOI: 10.1007/978-3-642-54568-9_14 (Zitiert auf S. 168).

[Bil12] N. Bilton. *Apple Loophole Gives Developers Access to Photos*. Pressemitteilung. The New York Times, Feb. 2012. URL: http://bits.blogs.nytimes.com/2012/02/28/tk-ios-gives-developers-access-to-photos-videos-location/ (Zitiert auf S. 28, 31, 32).

[BJM+16] D. Baloni, Jyotsna, M. Mandoli, P. Kathait und S. Nautiyal. „Comparision of two different andriod versions i. e Lollipop and Marshmallow, and also development of campsys mobile Andriod Application — An Overview". In: *International Journal Of Engineering And Computer Science* 5.5 (Mai 2016), Seiten 16518–16528. DOI: 10.18535/ijecs/v5i5.31 (Zitiert auf S. 62).

[BK11] R. C. Basole und J. Karla. „Entwicklung von Mobile-Platform-Ecosystem-Strukturen und -Strategien". In: *WIRTSCHAFTSINFORMATIK* 53.5 (Okt. 2011), Seiten 301–311. DOI: 10.1007/s11576-011-0286-y (Zitiert auf S. 50).

[BKK+15] M. Bitsaki, C. Koutras, G. Koutras, F. Leymann, B. Mitschang, C. Nikolaou, N. Siafakas, S. Strauch, N. Tzanakis und M. Wieland. „An Integrated mHealth Solution for Enhancing Patients' Health Online". In: *6th European Conference of the International Federation for Medical and Biological Engineering: MBEC 2014, 7-11 September 2014, Dubrovnik, Croatia*. Herausgegeben von I. Lacković und D. Vasic. Cham: Springer, 2015, Seiten 695–698. ISBN: 978-3-3191-1128-5. DOI: 10.1007/978-3-319-11128-5_173 (Zitiert auf S. 228).

[BKM+13] A. Bartel, J. Klein, M. Monperrus, K. Allix und Y. L. Traon. *Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation*. Technischer Bericht 1208.4536v2. Computing Research Repository, Okt. 2013 (Zitiert auf S. 147).

[BKvOS10] D. Barrera, H. G. Kayacik, P. C. van Oorschot und A. Somayaji. „A Methodology for Empirical Analysis of Permission-based Security Models and Its Application to Android". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. 2010, Seiten 73–84. DOI: 10.1145/1866307.1866317 (Zitiert auf S. 123).

[BMH10] F. Berg, F. Mack und P. Hilsbos. „Erschaffe Dein eigenes ‚mobile game'". Softwarepraktikum. Universität Stuttgart, Apr. 2010 (Zitiert auf S. 266, 269).

[BOZL13] Y. Ben Saied, A. Olivereau, D. Zeghlache und M. Laurent. „Trust Management System Design for the Internet of Things: A Context-aware and Multi-service Approach". In: *Computers and Security* 39 (Nov. 2013), Seiten 351–365. DOI: 10.1016/j.cose.2013.09.001 (Zitiert auf S. 38).

[BP15] A. Becker und M. Pant. *Android 5: Programmieren für Smartphones und Tablets*. 4. aktualisierte und erweiterte Auflage. Heidelberg: dpunkt.verlag, 2015. ISBN: 978-3-8649-0260-4 (Zitiert auf S. 61).

[BRD15] Bundesministeriums der Justiz und für Verbraucherschutz. *Bundesdatenschutzgesetz (BDSG)*. Bundesgesetz. Bundesrepublik Deutschland, Feb. 2015. URL: http://dejure.org/gesetze/BDSG (Zitiert auf S. 34, 41, 42).

[BRD16a] Bundesministeriums der Justiz und für Verbraucherschutz. *Gesetz über Urheberrecht und verwandte Schutzrechte (Urheberrechtsgesetz) – UrhG*. Bundesgesetz. Bundesrepublik Deutschland, Jan. 2016. URL: https://dejure.org/gesetze/UrhG (Zitiert auf S. 85, 169).

[BRD16b] Bundesministeriums der Justiz und für Verbraucherschutz. *Strafgesetzbuch (StGB)*. Bundesgesetz. Bundesrepublik Deutschland, Juli 2016. URL: https://dejure.org/gesetze/StGB (Zitiert auf S. 85, 169).

[BRSS11] A. R. Beresford, A. Rice, N. Skehin und R. Sohan. „MockDroid: Trading Privacy for Application Functionality on Smartphones". In: *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*. HotMobile '11. 2011, Seiten 49–54. DOI: 10.1145/2184489.2184500 (Zitiert auf S. 79, 174).

[BS03] A. Burak und T. Sharon. „Analyzing Usage of Location Based Services“. In: *Proceedings of the CHI '03 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '03. 2003, Seiten 970–971. DOI: 10.1145/765891.766100 (Zitiert auf S. 222).

[BS04] A. Burak und T. Sharon. „Usage Patterns of FriendZone: Mobile Location-based Community Services“. In: *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia*. MUM '04. 2004, Seiten 93–100. DOI: 10.1145/1052380.1052394 (Zitiert auf S. 222).

[BS09] A. Brodt und C. Stach. „Mobile ortsbasierte Browserspiele“. In: *Informatik 2009: Im Focus das Leben, Tagungsband der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 28.9.-2.10.2009, Lübeck*. Band 154. LNI. GI, Sep. 2009, Seiten 1902–1913. ISBN: 978-3-88579-248-2 (Zitiert auf S. 221, 222).

[BSI12] Bundesamt für Sicherheit in der Informationstechnik – BSI. *Leitfaden Informationssicherheit – IT-Grundschutz kompakt*. Bonn, Feb. 2012 (Zitiert auf S. 40).

[BSI16] Bundesamt für Sicherheit in der Informationstechnik – BSI. *Ihre Software sicher einrichten: Machen Sie Ihren Browser sicher*. Sicherheitsreport. BSI für Bürger, Juni 2016. URL: https://www.bsi-fuer-buerger.de/BSIFB/DE/Empfehlungen/EinrichtungSoftware/EinrichtungBrowser/Sicherheitsmassnahmen/SicherheitsCheck/sicherheitscheck_node.html (Zitiert auf S. 126).

[BSW07] J. Bethencourt, A. Sahai und B. Waters. „Ciphertext-Policy Attribute-Based Encryption“. In: *Proceedings of the 2007 IEEE Symposium on Security and Privacy*. SP '07. 2007, Seiten 321–334. DOI: 10.1109/SP.2007.11 (Zitiert auf S. 211).

[BV11] D. Barrera und P. Van Oorschot. „Secure Software Installation on Smartphones“. In: *IEEE Security and Privacy* 9.3 (Mai 2011), Seiten 42–48. DOI: 10.1109/MSP.2010.202 (Zitiert auf S. 63–65, 67, 69, 72).

[BVG83] Bundesverfassungsgericht. *BVerfGE 65, 1 – Volkszählung*. Urteil. Dez. 1983. URL: https://dejure.org/1983,1 (Zitiert auf S. 35).

[CB12] B. X. Chen und N. Bilton. *Et Tu, Google? Android Apps Can Also Secretly Copy Photos*. Pressemitteilung. The New York Times, März 2012. URL: http://bits.blogs.nytimes.com/2012/03/01/android-photos/ (Zitiert auf S. 31, 32).

[CBJP11] H. Choe, J. Baek, H. Jeong und S. Park. „MetaService: An Object Transfer Platform Between Android Applications“. In: *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*. RACS ’11. 2011, Seiten 56–60. DOI: 10.1145/2103380.2103391 (Zitiert auf S. 214).

[CCFZ12] M. Conti, B. Crispo, E. Fernandes und Y. Zhauniarovich. „CRêPE: A System for Enforcing Fine-Grained Context-Related Policies on Android“. In: *IEEE Transactions on Information Forensics and Security* 7.5 (Okt. 2012), Seiten 1426–1438. DOI: 10.1109/TIFS.2012.2204249 (Zitiert auf S. 79, 171).

[CDNO97] R. Canetti, C. Dwork, M. Naor und R. Ostrovsky. „Deniable Encryption“. In: *Advances in Cryptology — CRYPTO ’97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17-21, 1997. Proceedings*. Herausgegeben von B. S. Kaliski. Berlin, Heidelberg: Springer, 1997, Seiten 90–104. ISBN: 978-3-5406-9528-8. DOI: 10.1007/BFb0052229 (Zitiert auf S. 212).

[CEF+12] M. Chan, D. Estève, J.-Y. Fourniols, C. Escriba und E. Campo. „Smart Wearable Systems: Current Status and Future Challenges“. In: *Artificial Intelligence in Medicine* 56.3 (Nov. 2012), Seiten 137–156. DOI: 10.1016/j.artmed.2012.09.003 (Zitiert auf S. 121, 226).

[CFGW11] E. Chin, A. P. Felt, K. Greenwood und D. Wagner. „Analyzing Inter-application Communication in Android“. In: *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’11. 2011, Seiten 239–252. DOI: 10.1145/1999995.2000018 (Zitiert auf S. 60, 184, 252).

[CGKO06] R. Curtmola, J. Garay, S. Kamara und R. Ostrovsky. „Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions“. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS ’06. 2006, Seiten 79–88. DOI: 10.1145/1180405.1180417 (Zitiert auf S. 213).

[CH13] Y. Cherdantseva und J. Hilton. „A Reference Model of Information Assurance & Security“. In: *Proceedings of the 2013 International Conference on Availability, Reliability and Security*. ARES ’13. 2013, Seiten 546–555. DOI: 10.1109/ARES.2013.72 (Zitiert auf S. 41).

[CHC14] J. M. Chang, P.-C. Ho und T.-C. Chang. „Securing BYOD“. In: *IT Professional* 16.5 (2014), Seiten 9–11. DOI: 10.1109/MITP.2014.76 (Zitiert auf S. 232).

[CHY12] P. P. Chan, L. C. Hui und S. M. Yiu. „DroidChecker: Analyzing Android Applications for Capability Leak“. In: *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WISEC ’12. 2012, Seiten 125–136. DOI: 10.1145/2185448.2185466 (Zitiert auf S. 73, 86).

[CML15] B. Cici, A. Markopoulou und N. Laoutaris. „Designing an On-line Ride-sharing System“. In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS ’15. 2015, 60:1–60:4. DOI: 10.1145/2820783.2820850 (Zitiert auf S. 223).

[CNC11] M. Conti, V. T. N. Nguyen und B. Crispo. „CRêPE: Context-related Policy Enforcement for Android“. In: *Information Security: 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers*. Herausgegeben von M. Burmester, G. Tsudik, S. Magliveras und I. Ilić. Berlin, Heidelberg: Springer, 2011, Seiten 331–345. ISBN: 978-3-6421-8178-8. DOI: 10.1007/978-3-642-18178-8_29 (Zitiert auf S. 79, 171).

[CNSS13] Committee on National Security Systems. *CNSS Policy No. 15, Fact Sheet No. 1*. Datenblatt. NIST, Juni 2013. URL: http://csrc.nist.gov/groups/STM/cmvp/documents/CNSS15FS.pdf (Zitiert auf S. 195).

[CSDM12] N. Cipriani, C. Stach, O. Dörler und B. Mitschang. „NexusDSS – A System for Security Compliant Processing of Data Streams“. In: *Proceedings of the First International Conference on Data Technologies and Applications*. DATA ’12. **DATA 2012 Best Paper Award**. SciTePress, Juli 2012, Seiten 175–185. DOI: 10.5220/0004051401750185 (Zitiert auf S. 221, 235, 280).

[CWCZ15] B. Chang, Z. Wang, B. Chen und F. Zhang. „MobiPluto: File System Friendly Deniable Storage for Mobile Devices“. In: *Proceedings of the 31st Annual Computer Security Applications Conference*. ACSAC ’15. 2015, Seiten 381–390. DOI: 10.1145/2818000.2818046 (Zitiert auf S. 213).

[CWH+15] X. Cui, J. Wang, L. C. K. Hui, Z. Xie, T. Zeng und S. M. Yiu. „WeChecker: Efficient and Precise Detection of Privilege Escalation Vulnerabilities in Android Apps“. In: *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. WiSec ’15. 2015, 25:1–25:12. DOI: 10.1145/2766498.2766509 (Zitiert auf S. 86).

[CY16] Z. Cai und R. H. Yap. „Inferring the Detection Logic and Evaluating the Effectiveness of Android Anti-Virus Apps“. In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. CODASPY '16. 2016, Seiten 172–182. DOI: 10.1145/2857705.2857719 (Zitiert auf S. 210, 218–220).

[CZC11] M. Conti, I. Zachia-Zlatea und B. Crispo. „Mind How You Answer Me!: Transparently Authenticating the User of a Smartphone when Answering or Placing a Call“. In: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '11. 2011, Seiten 249–259. DOI: 10.1145/1966913.1966945 (Zitiert auf S. 171).

[CZG+15] P. Colp, J. Zhang, J. Gleeson, S. Suneja, E. de Lara, H. Raj, S. Saroiu und A. Wolman. „Protecting Data on Smartphones and Tablets from Memory Attacks“. In: *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '15. 2015, Seiten 177–189. DOI: 10.1145/2694344.2694380 (Zitiert auf S. 217).

[DA99] A. K. Dey und G. D. Abowd. *Towards a Better Understanding of Context and Context-Awareness*. Technischer Bericht. Georgia Tech GVU Center, 1999 (Zitiert auf S. 123).

[Dai11] D. A. Dai Zovi. *Apple iOS 4 Security Evaluation: Vulnerability Analysis and Data Encryption*. Whitepaper. Trail of Bits LLC, 2011. URL: http://media.blackhat.com/bh-us-11/DaiZovi/BH_US_11_DaiZovi_iOS_Security_WP.pdf (Zitiert auf S. 52).

[DC13] B. Davis und H. Chen. „RetroSkeleton: Retrofitting Android Apps“. In: *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '13. 2013, Seiten 181–192. DOI: 10.1145/2462456.2464462 (Zitiert auf S. 83, 84, 176).

[DDSW11] L. Davi, A. Dmitrienko, A.-R. Sadeghi und M. Winandy. „Privilege Escalation Attacks on Android“. In: *Information Security: 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers*. Herausgegeben von M. Burmester, G. Tsudik, S. Magliveras und I. Ilić. Berlin, Heidelberg: Springer, 2011, Seiten 346–360. ISBN: 978-3-6421-8178-8. DOI: 10.1007/978-3-642-18178-8_30 (Zitiert auf S. 168).

[Dey01] A. K. Dey. „Understanding and Using Context“. In: *Personal and Ubiquitous Computing* 5.1 (Feb. 2001), Seiten 4–7. DOI: 10.1007/s007790170019 (Zitiert auf S. 123).

[dHZ16] J. den Hartog und N. Zannone. „A Policy Framework for Data Fusion and Derived Data Control“. In: *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*. ABAC '16. 2016, Seiten 47–57. DOI: 10.1145/2875491.2875492 (Zitiert auf S. 92).

[DMS+16] S. Diesburg, C. Meyers, M. Stanovich, A.-I. A. Wang und G. Kuenning. „TrueErase: Leveraging an Auxiliary Data Path for Per-File Secure Deletion“. In: *ACM Transactions on Storage (TOS)* 12.4 (Aug. 2016), 18:1–18:37. DOI: 10.1145/2854882 (Zitiert auf S. 216).

[DN04] C. Dwork und K. Nissim. „Privacy-Preserving Datamining on Vertically Partitioned Databases“. In: *Advances in Cryptology – CRYPTO 2004: 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004. Proceedings*. Herausgegeben von M. Franklin. Berlin, Heidelberg: Springer, 2004, Seiten 528–544. ISBN: 978-3-5402-8628-8. DOI: 10.1007/978-3-540-28628-8_32 (Zitiert auf S. 231).

[Do05] N. Do. „How to Win at Tic-Tac-Toe“. In: *The Australian Mathematical Society Gazette* 32.3 (Juli 2005), Seiten 151–161 (Zitiert auf S. 222).

[DSK05] Die Beauftragten für Datenschutz und den Schutz der Privatsphäre. *Erklärung von Montreux: „Ein universelles Recht auf den Schutz personenbezogener Daten und der Privatsphäre unter Beachtung der Vielfalt in einer globalisierten Welt“*. Schlusserklärung. 27th International Conference of Data Protection und Privacy Commissioners, Sep. 2005. URL: http://www.lda.brandenburg.de/sixcms/detail.php/bb1.c.272605.de?_aria=ds (Zitiert auf S. 41).

[DSK11] G. Delac, M. Silic und J. Krolo. „Emerging Security Threats for Mobile Platforms“. In: *34th International Convention on Information and Communication Technology, Electronics and Microelectronics*. MIPRO '11. 2011, Seiten 1468–1473 (Zitiert auf S. 52).

[DSKC12] B. Davis, B. Sanders, A. Khodaverdian und H. Chen. „I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications“. In: *Proceeding of the 2012 IEEE Mobile Security Technologies*. MoST '12. 2012, 28:1–28:9 (Zitiert auf S. 84, 176).

[DWK81] G. I. Davida, D. L. Wells und J. B. Kam. „A Database Encryption System with Subkeys“. In: *ACM Transactions on Database Systems (TODS)* 6.2 (Juni 1981), Seiten 312–328. DOI: 10.1145/319566.319580 (Zitiert auf S. 213).

[Eas15] J. Eason. *An update on Eclipse Android Developer Tools*. Pressemitteilung. Google Inc., Jan. 2015. URL: http://android-developers.blogspot.de/2015/06/an-update-on-eclipse-android-developer.html (Zitiert auf S. 268).

[EBFK13] M. Egele, D. Brumley, Y. Fratantonio und C. Kruegel. „An Empirical Study of Cryptographic Misuse in Android Applications“. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. 2013, Seiten 73–84. DOI: 10.1145/2508859.2516693 (Zitiert auf S. 220).

[Eck13] C. Eckert. *IT-Sicherheit: Konzepte – Verfahren – Protokolle*. 8., aktualisierte und korrigierte Auflage. München: Oldenbourg, 2013. ISBN: 978-3-4867-2138-6 (Zitiert auf S. 40).

[EDL01] P. England, J. D. DeTreville und B. W. Lampson. *Digital rights management operating system*. Patent US6330670 B1. Microsoft, Dez. 2001 (Zitiert auf S. 88).

[EFW13] S. Egelman, A. P. Felt und D. Wagner. „Choice Architecture and Smartphone Privacy: There's a Price for That“. In: *The Economics of Information Security and Privacy*. Herausgegeben von R. Böhme. Berlin, Heidelberg: Springer, 2013. Kapitel 10, Seiten 211–236. ISBN: 978-3-6423-9498-0. DOI: 10.1007/978-3-642-39498-0_10 (Zitiert auf S. 66).

[EGC+10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel und A. N. Sheth. „TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones“. In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI '10. 2010, Seiten 393–407 (Zitiert auf S. 73, 86, 167, 179, 188).

[EGH+14] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel und A. N. Sheth. „TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones“. In: *ACM Transactions on Computer Systems (TOCS)* 32.2 (Juni 2014), 5:1–5:29. DOI: 10.1145/2619091 (Zitiert auf S. 73).

[Eil14] C. Eilers. *iOS Security: Sichere Apps für iPhone und iPad*. Frankfurt am Main: entwickler.press, 2014. ISBN: 978-3-8680-2101-1 (Zitiert auf S. 50–53).

[EKKV11] M. Egele, C. Kruegel, E. Kirda und G. Vigna. „PiOS : Detecting privacy leaks in iOS applications". In: *18th Annual Network and Distributed System Security Symposium*. NDSS '11. 2011, Seiten 1–15 (Zitiert auf S. 66).

[Ele14] N. Elenkov. *Android Security Internals: An In-Depth Guide to Android's Security Architecture*. San Francisco, CA, USA: No Starch Press, 2014. ISBN: 978-1-5932-7581-5 (Zitiert auf S. 58–61).

[EME+11] V. Etacheri, R. Marom, R. Elazari, G. Salitra und D. Aurbach. „Challenges in the development of advanced Li-ion batteries: a review". In: *Energy & Environmental Science* 4.9 (2011), Seiten 3243–3262. DOI: 10.1039/C1EE01598B (Zitiert auf S. 27).

[Ent13] Enterprise Mobility Solutions. *An Overview of the Samsung KNOX Platform*. Whitepaper. Samsung Electronics Co., Nov. 2013. URL: https://www2.samsungknox.com/en/blog/overview-samsung-knox-platform (Zitiert auf S. 217, 233).

[EOM09] W. Enck, M. Ongtang und P. McDaniel. „Understanding Android Security". In: *IEEE Security and Privacy* 7.1 (2009), Seiten 50–57. DOI: 10.1109/MSP.2009.26 (Zitiert auf S. 85, 186, 238).

[EOMC11] W. Enck, D. Octeau, P. McDaniel und S. Chaudhuri. „A Study of Android Application Security". In: *Proceedings of the 20th USENIX Conference on Security*. SEC '11. 2011, Seiten 21–36 (Zitiert auf S. 66).

[Erl04] Ú. Erlingsson. „The Inlined Reference Monitor Approach to Security Policy Enforcement". Dissertation. Cornell University, Jan. 2004 (Zitiert auf S. 83, 145).

[ES12] C. Eckert und C. Schneider. „Smart Mobile Apps: Enabler oder Risiko?" In: *Smart Mobile Apps: Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Herausgegeben von S. Verclas und C. Linnhoff-Popien. Berlin, Heidelberg: Springer, 2012. Kapitel 13, Seiten 193–207. ISBN: 978-3-6422-2259-7. DOI: 10.1007/978-3-642-22259-7_13 (Zitiert auf S. 229).

[EU12] Die Mitgliedstaaten der Europäischen Union. *Charta der Grundrechte der Europäischen Union*. Amtsblatt. Europäischen Union, Okt. 2012. URL: http://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:12012P/TXT (Zitiert auf S. 34, 41).

[EU16] Das Europäische Parlament und der Rat der Europäischen Union. *Verordnung (EU) 2016/679 des Europäischen Parlaments und der Rates vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung)*. Amtsblatt. Europäischen Union, Apr. 2016. URL: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679 (Zitiert auf S. 42).

[EU81] Die Mitgliedstaaten der Europäischen Union. *Übereinkommen zum Schutz des Menschen bei der automatischen Verarbeitung personenbezogener Daten*. Europäische Verträge. Europäischen Union, Jan. 1981. URL: http://www.coe.int/de/web/conventions/full-list/-/conventions/treaty/108 (Zitiert auf S. 34).

[EU95] Das Europäische Parlament und der Rat der Europäischen Union. *Richtlinie 95/46/EG des Europäischen Parlaments und des Rates vom 24. Oktober 1995 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten und zum freien Datenverkehr*. Amtsblatt. Europäischen Union, Okt. 1995. URL: http://eur-lex.europa.eu/eli/dir/1995/46/oj (Zitiert auf S. 41).

[eur16] eurodata. *So geht Zukunft: Smart Services sorgen für nahtlose Business-Prozesse – Was Unternehmen über Daten- und Prozessintegration wissen müssen, wenn Sie überleben wollen*. Whitepaper. eurodata AG, Sep. 2016. URL: http://edbic.de/ebook/ebook_edbic_eurodata_WEB_DE.pdf (Zitiert auf S. 233).

[FBJS12] E. Fragkaki, L. Bauer, L. Jia und D. Swasey. „Modeling and Enhancing Android's Permission System". In: *Computer Security – ESORICS 2012: 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*. Herausgegeben von S. Foresti, M. Yung und F. Martinelli. Berlin, Heidelberg: Springer, 2012, Seiten 1–18. ISBN: 978-3-6423-3167-1. DOI: 10.1007/978-3-642-33167-1_1 (Zitiert auf S. 79, 178).

[FBK+16] A. C. Franco da Silva, U. Breitenbücher, K. Képes, O. Kopp und F. Leymann. „OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker". In: *Proceedings of the 6th International Conference on the Internet of Things*. IoT '16. 2016, Seiten 181–182 (Zitiert auf S. 233).

[FCH+11] A. P. Felt, E. Chin, S. Hanna, D. Song und D. Wagner. „Android Permissions Demystified“. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. 2011, Seiten 627–638. DOI: 10.1145/2046707.2046779 (Zitiert auf S. 73, 244).

[FEF+12] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe und D. Wagner. „How to Ask for Permission“. In: *Proceedings of the 7th USENIX Conference on Hot Topics in Security*. HotSec '12. 2012, Seiten 1–6 (Zitiert auf S. 113, 237).

[Fer06] N. Ferguson. *AES-CBC + Elephant diffuser A Disk Encryption Algorithm for Windows Vista*. Technischer Bericht. Microsoft, Aug. 2006 (Zitiert auf S. 57).

[FEW12] A. P. Felt, S. Egelman und D. Wagner. „I've Got 99 Problems, but Vibration Ain't One: A Survey of Smartphone Users' Concerns“. In: *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM '12. 2012, Seiten 33–44. DOI: 10.1145/2381934.2381943 (Zitiert auf S. 62, 67, 260).

[FFC+11] A. P. Felt, M. Finifter, E. Chin, S. Hanna und D. Wagner. „A Survey of Mobile Malware in the Wild“. In: *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM '11. 2011, Seiten 3–14. DOI: 10.1145/2046614.2046618 (Zitiert auf S. 68).

[FGG+14] F. Freiling, R. Grimm, K.-E. Großpietsch, B. H. Keller, J. Mottok, I. Münch, K. Rannenberg und F. Saglietti. „Technische Sicherheit und Informationssicherheit“. In: *Informatik-Spektrum* 37.1 (2014), Seiten 14–24. DOI: 10.1007/s00287-013-0748-2 (Zitiert auf S. 35).

[FGS14] A. M. French, C. Guo und J. P. Shim. „Current Status, Issues, and Future of Bring Your Own Device (BYOD)“. In: *Communications of the Association for Information Systems – CAIS* 35.10 (Nov. 2014), Seiten 191–197 (Zitiert auf S. 232).

[FGW11] A. P. Felt, K. Greenwood und D. Wagner. „The Effectiveness of Application Permissions“. In: *Proceedings of the 2nd USENIX Conference on Web Application Development*. WebApps '11. 2011, Seiten 1–12 (Zitiert auf S. 30).

[FHDZ13] S. Fürst, M. Hoppe, F. K. Dang Huynh und J. Ziegler. „Evaluation of the Privacy Management Platform“. Projekt-INF. Universität Stuttgart, Aug. 2013 (Zitiert auf S. 261).

[FHE+12] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin und D. Wagner. „Android Permissions: User Attention, Comprehension, and Behavior“. In: *Proceedings of the Eighth Symposium on Usable Privacy and Security*. SOUPS '12. 2012, 3:1–3:14. DOI: 10.1145/2335356.2335360 (Zitiert auf S. 30, 67, 69, 73, 86, 186, 237, 242, 260).

[FHWM16] A. C. Franco da Silva, P. Hirmer, M. Wieland und B. Mitschang. „SitRS XT – Towards Near Real Time Situation Recognition“. In: *Journal of Information and Data Management (JIDM)* 7.1 (Apr. 2016), Seiten 4–17 (Zitiert auf S. 234).

[FMK+10] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan und D. Estrin. „Diversity in Smartphone Usage“. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. MobiSys '10. 2010, Seiten 179–194. DOI: 10.1145/1814433.1814453 (Zitiert auf S. 29, 31).

[Fur05] S. Furnell. „Handheld hazards: The rise of malware on mobile devices“. In: *Computer Fraud & Security* 2005.5 (Feb. 2005), Seiten 4–8. DOI: 10.1016/S1361-3723(05)70210-4 (Zitiert auf S. 31).

[FV13] P. Friess und O. Vermesan, Herausgeber. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. Aalborg: River Publishers, 2013. ISBN: 978-8-7929-8296-4 (Zitiert auf S. 233, 234).

[GAS10] M. A. M. Guimaraes, R. Austin und H. Said. „Database Forensics“. In: *Proceedings of the 2010 Information Security Curriculum Development Conference*. InfoSecCD '10. 2010, Seiten 62–65. DOI: 10.1145/1940941.1940958 (Zitiert auf S. 198).

[GBMP13] J. Gubbi, R. Buyya, S. Marusic und M. Palaniswami. „Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions“. In: *Future Generation Computer Systems* 29.7 (Sep. 2013), Seiten 1645–1660. DOI: 10.1016/j.future.2013.01.010 (Zitiert auf S. 233).

[GCCJ11] P. Gilbert, B.-G. Chun, L. P. Cox und J. Jung. „Vision: Automated Security Validation of Mobile Apps at App Markets“. In: *Proceedings of the Second International Workshop on Mobile Cloud Computing and Services*. MCS '11. 2011, Seiten 21–26. DOI: 10.1145/1999732.1999740 (Zitiert auf S. 86).

[Geb12] J. Gebauer. „Code Signing“. In: *Datenschutz und Datensicherheit – DuD* 36.9 (Sep. 2012), Seiten 684–684. DOI: 10.1007/s11623-012-0227-y (Zitiert auf S. 64).

[Gen09] C. Gentry. „Fully Homomorphic Encryption Using Ideal Lattices“. In: *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*. STOC ’09. 2009, Seiten 169–178. DOI: 10.1145/1536414.1536440 (Zitiert auf S. 213).

[GGH+13] O. Ganschar, S. Gerlach, M. Hämmerle, T. Krause und S. Schlund. *Produktionsarbeit der Zukunft – Industrie 4.0*. Herausgegeben von D. Spath. Stuttgart: Fraunhofer Verlag, 2013. ISBN: 978-3-8396-0570-7 (Zitiert auf S. 34).

[GH16] H. Grunert und A. Heuer. „Datenschutz im PArADISE“. In: *Datenbank-Spektrum* 16.2 (2016), Seiten 107–117. DOI: 10.1007/s13222-016-0216-7 (Zitiert auf S. 34).

[GHD+15] R. Gallo, P. Hongo, R. Dahab, L. C. Navarro, H. Kawakami, K. Galvão, G. Junqueira und L. Ribeiro. „Security and System Architecture: Comparison of Android Customizations“. In: *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. WiSec ’15. 2015, 12:1–12:6. DOI: 10.1145/2766498.2766519 (Zitiert auf S. 81).

[GHG10] T.-M. Grønli, J. Hansen und G. Ghinea. „Android vs Windows Mobile vs Java ME: A Comparative Study of Mobile Development Environments“. In: *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA ’10. 2010, 45:1–45:8. DOI: 10.1145/1839294.1839348 (Zitiert auf S. 49).

[GHGY14] T.-M. Grønli, J. Hansen, G. Ghinea und M. Younas. „Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS“. In: *Proceedings of the 2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. AINA ’14. 2014, Seiten 635–641. DOI: 10.1109/AINA.2014.78 (Zitiert auf S. 49).

[Gie16a] C. Giebler. „Erweiterungen für das eHealth-Spiel Secure CandyCastle“. Fachpraktikum. Universität Stuttgart, Okt. 2016 (Zitiert auf S. 229).

[Gie16b] C. Giebler. „Privatheit im Gesundheitsspiel Candy Castle“. Bachelorarbeit. Universität Stuttgart, Jan. 2016 (Zitiert auf S. 229).

[GJG+11] R. Geambasu, J. P. John, S. D. Gribble, T. Kohno und H. M. Levy. „Keypad: An Auditing File System for Theft-prone Devices“. In: *Proceedings of the Sixth Conference on Computer Systems*. EuroSys ’11. 2011, Seiten 1–16. DOI: 10.1145/1966445.1966447 (Zitiert auf S. 215).

[GKR16] P. Goel, L. Kulik und K. Ramamohanarao. „Privacy-Aware Dynamic Ride Sharing“. In: *ACM Transactions on Spatial Algorithms and Systems* 2.1 (Apr. 2016), 4:1–4:41. DOI: 10.1145/2845080 (Zitiert auf S. 223).

[GM14] J. Götzfried und T. Müller. „Analysing Android’s Full Disk Encryption Feature“. In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)* 5.1 (März 2014), Seiten 84–100 (Zitiert auf S. 61).

[GN14] P. Getzmann und P. Nowak. *Das neue Windows Phone 8.1 für den Business-Einsatz*. Pressemitteilung. Microsoft, Apr. 2014. URL: https://www.microsoft.com/germany/techwiese/know-how/entwickeln-fuer-windows-phone-8.1/das-neue-windows-phone-8.1-fuer-den-business-einsatz.aspx (Zitiert auf S. 57).

[GNSM12] C. Gröger, F. Niedermann, H. Schwarz und B. Mitschang. „Supporting Manufacturing Design by Analytics, Continuous Collaborative Process Improvement Enabled by the Advanced Manufacturing Analytics Platform“. In: *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design*. CSCWD ’12. Mai 2012, Seiten 793–799. DOI: 10.1109/CSCWD.2012.6221911 (Zitiert auf S. 230).

[Goo16] D. Goodin. *Android’s full-disk encryption just got much weaker—here’s why: Unlike Apple’s iOS, Android is vulnerable to several key-extraction techniques.* Pressemitteilung. Ars Technica, Jan. 2016. URL: http://arstechnica.com/security/2016/07/androids-full-disk-encryption-just-got-much-weaker-heres-why/ (Zitiert auf S. 61).

[Got12] G. Goth. „Mobile Security Issues Come to the Forefront“. In: *IEEE Internet Computing* 16.3 (Mai 2012), Seiten 7–9. DOI: 10.1109/MIC.2012.54 (Zitiert auf S. 67).

[GPSW06] V. Goyal, O. Pandey, A. Sahai und B. Waters. „Attribute-based Encryption for Fine-grained Access Control of Encrypted Data“. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS ’06. 2006, Seiten 89–98. DOI: 10.1145/1180405.1180418 (Zitiert auf S. 211).

[GRB10] S. Ghelawat, K. Radke und M. Brereton. „Interaction, Privacy and Profiling Considerations in Local Mobile Social Software: A Prototype Agile Ride Share System“. In: *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*. OZCHI ’10. 2010, Seiten 376–379. DOI: 10.1145/1952222.1952307 (Zitiert auf S. 223).

[Grö15] C. Gröger. „Advanced Manufacturing Analytics – Datengetriebene Optimierung von Fertigungsprozessen“. Dissertation. Universität Stuttgart, Sep. 2015 (Zitiert auf S. 230).

[GS14] C. Gröger und C. Stach. „The Mobile Manufacturing Dashboard“. In: *Proceedings of the 2014 IEEE International Conference on Pervasive Computing and Communications Workshops*. PerCom ’14. IEEE, März 2014, Seiten 138–140. DOI: 10.1109/PerComW.2014.6815180 (Zitiert auf S. 221, 230, 358).

[GS17] C. Giebler und C. Stach. „Datenschutzmechanismen für Gesundheitsspiele am Beispiel von Secure Candy Castle“. In: *Tagungsband der 15. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web*. BTW ’17. GI, März 2017, Seiten 311–320 (Zitiert auf S. 221, 227).

[GSG14] B. M. Gaff, H. E. Sussman und J. Geetter. „Privacy and Big Data“. In: *Computer* 47.6 (Juni 2014), Seiten 7–9. DOI: 10.1109/MC.2014.161 (Zitiert auf S. 183).

[GSMW16] C. Gröger, C. Stach, B. Mitschang und E. Westkämper. „A mobile dashboard for analytics-based information provisioning on the shop floor“. In: *International Journal of Computer Integrated Manufacturing* (Mai 2016), Seiten 1–20. ISSN: 1362-3052. DOI: 10.1080/0951192X.2016.1187292 (Zitiert auf S. 221, 230).

[GVR15] P. Gerber, M. Volkamer und K. Renaud. „Usability Versus Privacy Instead of Usable Privacy: Google’s Balancing Act Between Usability and Privacy“. In: *ACM SIGCAS Computers and Society* 45.1 (Feb. 2015), Seiten 16–21. DOI: 10.1145/2738210.2738214 (Zitiert auf S. 31).

[GWB10] V. Gonçalves, N. Walravens und P. Ballon. „„How About an App Store?“ Enablers and Constraints in Platform Strategies for Mobile Network Operators“. In: *Proceedings of the 2010 Ninth International Conference on Mobile Business / 2010 Ninth Global Mobility Roundtable*. ICMB-GMR ’10. 2010, Seiten 66–73. DOI: 10.1109/ICMB-GMR.2010.41 (Zitiert auf S. 68).

[GZWJ12] M. Grace, Y. Zhou, Z. Wang und X. Jiang. „Systematic Detection of Capability Leaks in Stock Android Smartphones“. In: *Proceedings of the 2012 Network and Distributed System Security Symposium*. NDSS ’12. 2012, 7/5:1–7/5:15 (Zitiert auf S. 173).

[GZZ+12] M. Grace, Y. Zhou, Q. Zhang, S. Zou und X. Jiang. „RiskRanker: Scalable and Accurate Zero-day Android Malware Detection“. In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’12. 2012, Seiten 281–294. DOI: 10.1145/2307636.2307663 (Zitiert auf S. 36).

[HA16] J. Hallett und D. Aspinall. „AppPAL for Android“. In: *Engineering Secure Software and Systems: 8th International Symposium, ESSoS 2016, London, UK, April 6-8, 2016. Proceedings*. Herausgegeben von J. Caballero, E. Bodden und E. Athanasopoulos. Cham: Springer, 2016, Seiten 216–232. ISBN: 978-3-3193-0806-7. DOI: 10.1007/978-3-319-30806-7_14 (Zitiert auf S. 87).

[HAB+09] E. Hammer-Lahav, M. Atwood, D. Balfanz, D. Bounds, R. M. Conlan, B. Cook, L. Culver, B. de Medeiros, B. Eaton, K. Elliott-McCrea, L. Halff, B. Laurie, C. Messina, J. Panzer, S. Quigley, D. Recordon, E. Sandler, J. Sergent, T. Sieling, B. Slesinsky und A. Smith. *OAuth Core 1.0 Revision A*. Spezifikation. OAuth Core Workgroup, Juni 2009. URL: https://oauth.net/core/1.0a/ (Zitiert auf S. 217).

[Hac07] D. K. Hackborn. *OpenBinder*. Architektur- und Schnittstellenbeschreibung. Palm, Inc., Okt. 2007. URL: http://www.angryredplanet.com/~hackbod/openbinder/docs/html/index.html (Zitiert auf S. 128).

[Ham15] K. Hampe. *44 Millionen Deutsche nutzen ein Smartphone*. Pressemitteilung. Bitkom Research, März 2015. URL: http://www.bitkom-research.de/epages/63742557.sf/de_DE/?ObjectPath=/Shops/63742557/Categories/Presse/Pressearchiv_2015/44_Millionen_Deutsche_nutzen_ein_Smartphone (Zitiert auf S. 29).

[HD10] G. Hogben und M. Dekker. *Smartphones: Information security risks, opportunities and recommendations for users*. Sicherheitsreport. European Union Agency for Network und Information Security (ENISA), Dez. 2010. URL: https://www.enisa.europa.eu/publications/smartphones-information-security-risks-opportunities-and-recommendations-for-users (Zitiert auf S. 63, 65).

[HDE16] M. Harbach, A. De Luca und S. Egelman. „The Anatomy of Smartphone Unlocking: A Field Study of Android Lock Screens“. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. 2016, Seiten 4806–4817. DOI: 10.1145/2858036.2858267 (Zitiert auf S. 239, 241).

[Hei15] P. Heidkamp. *Deutschland fehlt es an digitalem Selbstbewusstsein: Unzureichender Breitbandausbau ist ein Wettbewerbsnachteil für den Standort Deutschland*. Pressemitteilung. KPMG International, Nov. 2015. URL: https://klardenker.kpmg.de/deutschland-fehlt-es-an-digitalem-selbstbewusstsein/ (Zitiert auf S. 35, 37).

[Her13] D. Hernie. *Windows Phone 8 Security deep dive*. MobCom Workshop Vortrag. Microsoft Corporation, Feb. 2013. URL: https://www.msec.be/mobcom/ws2013/ (Zitiert auf S. 54, 55, 65).

[HGKM14] E. Hoos, C. Gröger, S. Kramer und B. Mitschang. „Improving Business Processes Through Mobile Apps – An Analysis Framework to Identify Value-added App Usage Scenarios“. In: *Proceedings of the 16th International Conference on Enterprise Information Systems*. ICEIS '14. 2014, Seiten 71–82. DOI: 10.5220/0004897100710082 (Zitiert auf S. 229).

[HGKM15] E. Hoos, C. Gröger, S. Kramer und B. Mitschang. „ValueApping: An Analysis Method to Identify Value-Adding Mobile Enterprise Apps in Business Processes“. In: *Enterprise Information Systems: 16th International Conference, ICEIS 2014, Lisbon, Portugal, April 27-30, 2014, Revised Selected Papers*. Herausgegeben von J. Cordeiro, S. Hammoudi, L. Maciaszek, O. Camp und J. Filipe. Cham: Springer, 2015, Seiten 222–243. ISBN: 978-3-3192-2348-3. DOI: 10.1007/978-3-319-22348-3_13 (Zitiert auf S. 229).

[HGM15] E. Hoos, C. Gröger und B. Mitschang. „Mobile Apps in Engineering: A Process-Driven Analysis of Business Potentials and Technical Challenges“. In: *Procedia CIRP – 9th CIRP Conference on Intelligent Computation in Manufacturing Engineering – CIRP ICME '14* 33 (2015), Seiten 17–22. DOI: 10.1016/j.procir.2015.06.005 (Zitiert auf S. 229).

[HHJ+11] P. Hornyack, S. Han, J. Jung, S. Schechter und D. Wetherall. „These Aren't the Droids You're Looking for: Retrofitting Android to Protect Data from Imperious Applications“. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. 2011, Seiten 639–652. DOI: 10.1145/2046707.2046780 (Zitiert auf S. 78, 79, 167).

[Hir14] H. Hirsch-Kreinsen. *Wandel von Produktionsarbeit – „Industrie 4.0“*. Technischer Bericht 38. Technische Universität Dortmund, Jan. 2014 (Zitiert auf S. 229).

[HJ08] K. W. Hamlen und M. Jones. „Aspect-oriented In-lined Reference Monitors“. In: *Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*. PLAS '08. 2008, Seiten 11–20. DOI: 10.1145/1375696.1375699 (Zitiert auf S. 168).

[HJS12] K. W. Hamlen, M. M. Jones und M. Sridhar. „Aspect-Oriented Runtime Monitor Certification“. In: *Tools and Algorithms for the Construction and Analysis of Systems: 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*. Herausgegeben von C. Flanagan und B. König. Berlin, Heidelberg: Springer, 2012, Seiten 126–140. ISBN: 978-3-6422-8756-5. DOI: 10.1007/978-3-642-28756-5_10 (Zitiert auf S. 168).

[Hol14] F. W. Holliday. „IT-Sicherheitsmanagement – Zeit für einen Paradigmenwechsel“. In: *Informatik-Spektrum* 37.1 (2014), Seiten 25–27. DOI: 10.1007/s00287-013-0726-8 (Zitiert auf S. 42).

[Hoo12] A. Hoog. *Android Forensik: Datenrecherche, Analyse und mobile Sicherheit bei Android*. Haar: Franzis Verlag GmbH, 2012. ISBN: 978-3-6456-0210-5 (Zitiert auf S. 60).

[HS11] A. Hoog und K. Strzempka. *iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices*. Waltham, MA, USA: Syngress Publishing, 2011. ISBN: 978-1-5974-9659-9 (Zitiert auf S. 53).

[HS16] M. Hüffmeyer und U. Schreier. „RestACL: An Access Control Language for RESTful Services“. In: *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*. ABAC '16. 2016, Seiten 58–67. DOI: 10.1145/2875491.2875494 (Zitiert auf S. 280).

[HSB+05] R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben und J. Reitsma. „Context Sensitive Access Control“. In: *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*. SACMAT '05. 2005, Seiten 111–119. DOI: 10.1145/1063979.1064000 (Zitiert auf S. 123).

[HSD13] H. Hao, V. Singh und W. Du. „On the Effectiveness of API-level Access Control Using Bytecode Rewriting in Android“. In: *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. ASIA CCS '13. 2013, Seiten 25–36. DOI: 10.1145/2484313.2484317 (Zitiert auf S. 84, 145).

[HWS+15] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher und F. Leymann. „SitRS – A Situation Recognition Service based on Modeling and Executing Situation Templates“. In: *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*. SummerSOC '15. 2015, Seiten 113–127 (Zitiert auf S. 234).

[ISO09] ISO. *Information technology – Security techniques – Information security management systems – Overview and vocabulary*. ISO-Standard ISO/IEC 27000:2009. Geneva, Switzerland: International Organization for Standardization, 2009 (Zitiert auf S. 39).

[ISO13a] ISO. *Country Codes*. ISO-Standard ISO 3166. Geneva, Switzerland: International Organization for Standardization, 2013 (Zitiert auf S. 96).

[ISO13b] ISO. *Information technology – Security techniques – Information security management systems – Requirements*. ISO-Standard ISO/IEC 27001:2013. Geneva, Switzerland: International Organization for Standardization, 2013 (Zitiert auf S. 39).

[ISO14] ISO. *ISO/IEC/IEEE Health informatics–Personal health device communication–Part 20601: Application profile–Optimized exchange protocol*. ISO-Standard ISO/IEEE 11073-20601:2014. Geneva, Switzerland: International Organization for Standardization, 2014 (Zitiert auf S. 226).

[Jac16] G. Jacobson. „Schlüsselverteilung für Mobilgeräte“. In: *Datenschutz und Datensicherheit - DuD* 40.4 (2016), Seiten 217–221. DOI: 10.1007/s11623-016-0581-2 (Zitiert auf S. 56).

[Jaz14] N. Jazdi. „Cyber physical systems in the context of Industry 4.0“. In: *Proceedings of the 2014 IEEE International Conference on Automation, Quality and Testing, Robotics*. AQTR '14. Mai 2014, Seiten 1–3. DOI: 10.1109/AQTR.2014.6857843 (Zitiert auf S. 233).

[JBA12] C. Jarabek, D. Barrera und J. Aycock. „ThinAV: Truly Lightweight Mobile Cloud-based Anti-malware“. In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACSAC ’12. 2012, Seiten 209–218. DOI: 10.1145/2420950.2420983 (Zitiert auf S. 218, 278).

[JBBL12] S. Jeon, J. Bang, K. Byun und S. Lee. „A Recovery Method of Deleted Record for SQLite Database“. In: *Personal and Ubiquitous Computing* 16.6 (Aug. 2012), Seiten 707–715. DOI: 10.1007/s00779-011-0428-7 (Zitiert auf S. 198).

[Jen14] K. Jendrian. „Der Standard ISO/IEC 27001:2013“. In: *Datenschutz und Datensicherheit – DuD* 38.8 (Aug. 2014), Seiten 552–557. DOI: 10.1007/s11623-014-0182-x (Zitiert auf S. 39).

[JM11] E. Jovanov und A. Milenkovic. „Body Area Networks for Ubiquitous Healthcare Applications: Opportunities and Challenges“. In: *Journal of Medical Systems* 35.5 (Okt. 2011), Seiten 1245–1254. DOI: 10.1007/s10916-011-9661-x (Zitiert auf S. 226).

[JMK13] M. E. Joorabchi, A. Mesbah und P. Kruchten. „Real Challenges in Mobile App Development“. In: *Proceedings of the 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 2013, Seiten 15–24. DOI: 10.1109/ESEM.2013.9 (Zitiert auf S. 30).

[JMV+12] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster und T. Millstein. „Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications“. In: *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM ’12. 2012, Seiten 3–14. DOI: 10.1145/2381934.2381938 (Zitiert auf S. 84, 172).

[Joy14] J. Joy. *Penetration Testing Windows Phone Applications*. Vortrag. Ernst & Young Global Limited, Juni 2014. URL: http://de.slideshare.net/JewelJoy/windows-phone-application-penetration-testing (Zitiert auf S. 55, 56).

[JPZ06] N. Joukov, H. Papaxenopoulos und E. Zadok. „Secure Deletion Myths, Issues, and Solutions“. In: *Proceedings of the Second ACM Workshop on Storage Security and Survivability*. StorageSS ’06. 2006, Seiten 61–66. DOI: 10.1145/1179559.1179571 (Zitiert auf S. 198).

[JSCK13] H. Jin, G. Saldamli, R. Chow und B. P. Knijnenburg. „Recommendations-based Location Privacy Control“. In: *2013 IEEE International Conference on Pervasive Computing and Communications Workshops*. PERCOM Workshops ’13. 2013, Seiten 401–404. DOI: 10.1109/PerComW.2013.6529526 (Zitiert auf S. 87).

[Jul13] M. d. O. Jullien. „Candy Castle : Um Jogo Sério para Pacientes com Diabetes“. Bachelorarbeit. Universidade Federal do Rio Grande do Sul, Dez. 2013 (Zitiert auf S. 226).

[Kas13] M. Kassner. *Android flashlight app tracks users via GPS, FTC says hold on*. Pressemitteilung. TechRepublic, Dez. 2013. URL: http://www.techrepublic.com/blog/it-security/why-does-an-android-flashlight-app-need-gps-permission/ (Zitiert auf S. 32, 33).

[Kel14] G. Kelly. *97% Of Mobile Malware Is On Android. This Is The Easy Way You Stay Safe*. Sicherheitsreport. Forbes Media, März 2014. URL: http://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-malware-is-on-android-this-is-the-easy-way-you-stay-safe/ (Zitiert auf S. 53).

[KFJ01] L. Kagal, T. Finin und A. Joshi. „Trust-Based Security in Pervasive Computing Environments“. In: *Computer* 34.12 (Dez. 2001), Seiten 154–157. DOI: 10.1109/2.970591 (Zitiert auf S. 38).

[KGT+15] K. Kuznetsov, A. Gorla, I. Tavecchia, F. Groß und A. Zeller. „Mining Android Apps for Anomalies“. In: *The Art and Science of Analyzing Software Data*. Herausgegeben von C. Bird, T. Menzies und T. Zimmermann. Boston, MA, USA: Morgan Kaufmann, 2015. Kapitel 10, Seiten 257–283. ISBN: 978-0-1241-1519-4. DOI: 10.1016/B978-0-12-411519-4.00010-0 (Zitiert auf S. 210).

[Kim10] K. Kimbler. „App Store Strategies for Service Providers“. In: *2010 14th International Conference on Intelligence in Next Generation Networks*. ICIN ’10. 2010, Seiten 1–5. DOI: 10.1109/ICIN.2010.5640947 (Zitiert auf S. 68).

[KK11] K. Karaoglanoglou und H. Karatza. „Resource Discovery in a Grid System: Directing Requests to Trustworthy Virtual Organizations Based on Global Trust Values“. In: *Journal of Systems and Software* 84.3 (März 2011), Seiten 465–478. DOI: 10.1016/j.jss.2010.10.043 (Zitiert auf S. 38).

[KKKH15] J. Kang, D. Kim, H. Kim und J. H. Huh. „Analyzing Unnecessary Permissions Requested by Android Apps Based on Users' Opinions". In: *Information Security Applications: 15th International Workshop, WISA 2014, Jeju Island, Korea, August 25-27, 2014. Revised Selected Papers*. Herausgegeben von K.-H. Rhee und H. J. Yi. Cham: Springer, 2015, Seiten 68–79. ISBN: 978-3-3191-5087-1. DOI: 10.1007/978-3-319-15087-1_6 (Zitiert auf S. 172).

[KKR10] A. Kaminsky, M. Kurdziel und S. Radziszowski. „An Overview of Cryptanalysis Research for the Advanced Encryption Standard". In: *Proceedings of the 2010 IEEE Military Communications Conference*. MILCOM '10. 2010, Seiten 1310–1316. DOI: 10.1109/MILCOM.2010.5680130 (Zitiert auf S. 248).

[Kli15] S. Klipper. *Information Security Risk Management: Risikomanagement mit ISO/IEC 27001, 27005 und 31010*. Herausgegeben von P. Hohl. 2., überarbeitete Auflage. Wiesbaden: Springer, 2015. ISBN: 978-3-6580-8773-9. DOI: 10.1007/978-3-658-08774-6 (Zitiert auf S. 39).

[KLW11] H. Kagermann, W.-D. Lukas und W. Wahlster. „Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution". In: *VDI nachrichten* 13 (Apr. 2011), Seiten 2–2 (Zitiert auf S. 229).

[KM11] M. Knöll und M. Moar. „On the Importance of Locations in Therapeutic Serious Games: Review on current health games and how they make use of the urban landscape". In: *Proceedings of the 5th International Conference on Pervasive Computing Technologies for Healthcare*. PervasiveHealth '11. 2011, Seiten 538–545. DOI: 10.4108/icst.pervasivehealth.2011.246013 (Zitiert auf S. 247).

[KNK+12] P. Kodeswaran, V. Nandakumar, S. Kapoor, P. Kamaraju, A. Joshi und S. Mukherjea. „Securing Enterprise Data on Smartphones Using Run Time Information Flow Control". In: *Proceedings of the 2012 IEEE 13th International Conference on Mobile Data Management*. 2012, Seiten 300–305. DOI: 10.1109/MDM.2012.50 (Zitiert auf S. 86).

[Knö10a] M. Knöll. „"On the Top of High Towers . . ." Discussing Locations in a Mobile Health Game for Diabetics". In: *Proceedings of the 2010 IADIS International Conference Game and Entertainment Technologies*. MCCSIS '10. 2010, Seiten 61–68 (Zitiert auf S. 225).

[Knö10b] M. Knöll. „Pervasive Health Games“. In: *Serious Game Design and Development: Technologies for Training and Learning*. Herausgegeben von J. Cannon-Bowers und C. Bowers. Hershey, PA, USA: IGI Global, 2010. Kapitel 14, Seiten 260–269. ISBN: 978-1-6152-0739-8. DOI: 10.4018/978-1-61520-739-8.ch014 (Zitiert auf S. 225).

[KNRR16] S. Kumar, L. G. Nagaraj, P. Rawal und P. Rohilla. „Sideloading and Windows App Certification Kit“. In: *Windows 10 Development Recipes: A Problem-Solution Approach in HTML and JavaScript*. Berkeley, CA, USA: Apress, 2016. Kapitel 17, Seiten 327–345. ISBN: 978-1-4842-0719-2. DOI: 10.1007/978-1-4842-0719-2_17 (Zitiert auf S. 68).

[KP13a] F. Kelbert und A. Pretschner. „Data Usage Control Enforcement in Distributed Systems“. In: *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*. CODASPY '13. 2013, Seiten 71–82. DOI: 10.1145/2435349.2435358 (Zitiert auf S. 92).

[KP13b] P. Kumari und A. Pretschner. „Model-Based Usage Control Policy Derivation“. In: *Engineering Secure Software and Systems: 5th International Symposium, ESSoS 2013, Paris, France, February 27 - March 1, 2013. Proceedings*. Herausgegeben von J. Jürjens, B. Livshits und R. Scandariato. Berlin, Heidelberg: Springer, 2013, Seiten 58–74. ISBN: 978-3-6423-6563-8. DOI: 10.1007/978-3-642-36563-8_5 (Zitiert auf S. 92).

[KP14] F. Kelbert und A. Pretschner. „Decentralized Distributed Data Usage Control“. In: *Cryptology and Network Security: 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*. Herausgegeben von D. Gritzalis, A. Kiayias und I. Askoxylakis. Cham: Springer, 2014, Seiten 353–369. ISBN: 978-3-3191-2280-9. DOI: 10.1007/978-3-319-12280-9_23 (Zitiert auf S. 92).

[KR11] C. Kurz und F. Rieger. *Die Datenfresser: Wie Internetfirmen und Staat sich unsere persönlichen Daten einverleiben und wie wir die Kontrolle darüber zurückerlangen*. Frankfurt am Main: S. Fischer Verlag GmbH, 2011. ISBN: 978-3-1004-8518-2 (Zitiert auf S. 28).

[Küp05] A. Küpper. *Location-based Services : Fundamentals and Operation*. West Sussex: John Wiley & Sons Ltd., 2005. ISBN: 978-0-4700-9231-6 (Zitiert auf S. 222).

[KW16] U. Kanonov und A. Wool. „Secure Containers in Android: The Samsung KNOX Case Study“. In: *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM ’16. 2016, Seiten 3–12. DOI: 10.1145/2994459.2994470 (Zitiert auf S. 232).

[KWKH14] M. Knöll, C. Wagner, D. Köse und M. Holder. „Wo die Monster leben – Welche Orte und Begleitung haben Einfluss auf das Blutzuckermessen und können zur Entwicklung von Serious Games für Typ-1-Diabetiker beitragen“. In: *Diabetologie und Stoffwechsel* S 01.9 (Juli 2014), Seiten 303–303. DOI: 10.1055/s-0034-1375160 (Zitiert auf S. 225).

[LBK14a] I. Litou, I. Boutsis und V. Kalogeraki. „Efficient Dissemination of Emergency Information using a Social Network“. In: *Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference*. EDBT/ICDT-WS ’16. 2014, Seiten 347–354 (Zitiert auf S. 279).

[LBK14b] I. Litou, I. Boutsis und V. Kalogeraki. „Using Location-Based Social Networks for Time-Constrained Information Dissemination“. In: *Proceedings of the 2014 IEEE 15th International Conference on Mobile Data Management*. MDM ’14. 2014, Seiten 162–171. DOI: 10.1109/MDM.2014.26 (Zitiert auf S. 279).

[Lea11] N. Leavitt. „Mobile Security: Finally a Serious Problem?“ In: *Computer* 44.6 (Juni 2011), Seiten 11–14. DOI: 10.1109/MC.2011.184 (Zitiert auf S. 28, 67).

[LEPM12] I. Leontiadis, C. Efstratiou, M. Picone und C. Mascolo. „Don’t kill my ads! Balancing Privacy in an Ad-Supported Mobile Application Market“. In: *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. HotMobile ’12. 2012, Seiten 21–26. DOI: 10.1145/2162081.2162084 (Zitiert auf S. 66, 183, 262).

[LGL14] J. Larson, J. Glanz und A. W. Lehren. *Spy Agencies Probe Angry Birds and Other Apps for Personal Data*. Pressemitteilung. ProPublica Inc., Jan. 2014. URL: https://www.propublica.org/article/spy-agencies-probe-angry-birds-and-other-apps-for-personal-data (Zitiert auf S. 32, 33).

[LHC+08] J. Lee, J. Heo, Y. Cho, J. Hong und S. Y. Shin. „Secure Deletion for NAND Flash File System“. In: *Proceedings of the 2008 ACM Symposium on Applied Computing*. SAC ’08. 2008, Seiten 1710–1714. DOI: 10.1145/1363686.1364093 (Zitiert auf S. 199).

[LK06] J. T. Lehikoinen und A. Kaikkonen. „PePe Field Study: Constructing Meanings for Locations in the Context of Mobile Presence“. In: *Proceedings of the 8th Conference on Human-computer Interaction with Mobile Devices and Services*. MobileHCI '06. 2006, Seiten 53–60. DOI: 10.1145/1152215.1152228 (Zitiert auf S. 222).

[LKBG16] I. Litou, V. Kalogeraki, I. Boutsis und D. Gunopulos. „Real-Time and Cost-Effective Limitation of Misinformation Propagation“. In: *Proceedings of the 2016 IEEE 17th International Conference on Mobile Data Management*. MDM '16. 2016, Seiten 158–163. DOI: 10.1109/MDM.2016.33 (Zitiert auf S. 279).

[LMS+14] S. Lortz, H. Mantel, A. Starostin, T. Bähr, D. Schneider und A. Weber. „Cassandra: Towards a Certifying App Store for Android“. In: *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. SPSM '14. 2014, Seiten 93–104. DOI: 10.1145/2666620.2666631 (Zitiert auf S. 210).

[LP11] E. Lovat und A. Pretschner. „Data-centric Multi-layer Usage Control Enforcement: A Social Network Example“. In: *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*. SACMAT '11. 2011, Seiten 151–152. DOI: 10.1145/1998441.1998467 (Zitiert auf S. 92).

[LPZ14] M. Loureiro, F. Prager und H. Zimmermann. „Design of an Online Marketplace for Mobile Applications“. Projekt-INF. Universität Stuttgart, Mai 2014 (Zitiert auf S. 278).

[LRN15] R. Llamas, R. Reith und K. Nagamine. *Smartphone OS Market Share, 2015 Q2*. Pressemitteilung. IDC Research, Inc., Aug. 2015. URL: http://www.idc.com/prodserv/smartphone-os-market-share.jsp (Zitiert auf S. 30, 43).

[Luc01] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 978-0-2017-2789-0 (Zitiert auf S. 234).

[Luc15] C. Lucien. *Sensors are vital to smartphones today*. Pressemitteilung. RCR Wireless News, März 2015. URL: http://www.rcrwireless.com/20150302/opinion/reader-forum-sensor-adoption-in-smartphones-tag10 (Zitiert auf S. 28, 29).

[Lud13] A. Ludwig. *The value of openness in Android security*. Pressemitteilung. Google Inc., Dez. 2013. URL: https://android.googleblog.com/2013/12/the-value-of-openness-in-android.html (Zitiert auf S. 91).

[Man12] S. Mansfield-Devine. „Android malware and mitigations“. In: *Network Security* 2012.11 (Nov. 2012), Seiten 12–20. DOI: 10.1016/S1353-4858(12)70104-6 (Zitiert auf S. 58).

[May15] F. A. Mayer. „Ein sicherer Datencontainer für die Cloud“. Diplomarbeit. Universität Stuttgart, Dez. 2015 (Zitiert auf S. 225, 235, 280).

[MBM12] D. Mukherjee, S. Banerjee und P. Misra. „Ad-hoc Ride Sharing Application Using Continuous SPARQL Queries“. In: *Proceedings of the 21st International Conference on World Wide Web*. WWW '12 Companion. 2012, Seiten 579–580. DOI: 10.1145/2187980.2188136 (Zitiert auf S. 223).

[MBP15] S. Mutti, E. Bacis und S. Paraboschi. „SeSQLite: Security Enhanced SQLite: Mandatory Access Control for Android Databases“. In: *Proceedings of the 31st Annual Computer Security Applications Conference*. ACSAC '15. 2015, Seiten 411–420. DOI: 10.1145/2818000.2818041 (Zitiert auf S. 215).

[MCSB08] P. Mazzoleni, B. Crispo, S. Sivasubramanian und E. Bertino. „XACML Policy Integration Algorithms“. In: *ACM Transactions on Information and System Security (TISSEC)* 11.1 (Feb. 2008), 4:1–4:29. DOI: 10.1145/1330295.1330299 (Zitiert auf S. 171).

[MDTG11] A. Mylonas, S. Dritsas, B. Tsoumas und D. Gritzalis. „Smartphone security evaluation – the malware attack case“. In: *Proceedings of the 2011 International Conference on Security and Cryptography*. SECRYPT '11. 2011, Seiten 25–36 (Zitiert auf S. 64–66).

[MDTG12] A. Mylonas, S. Dritsas, B. Tsoumas und D. Gritzalis. „On the Feasibility of Malware Attacks in Smartphone Platforms“. In: *E-Business and Telecommunications: International Joint Conference, ICETE 2011, Seville, Spain, July 18-21, 2011, Revised Selected Papers*. Herausgegeben von M. S. Obaidat, J. L. Sevillano und J. Filipe. Berlin, Heidelberg: Springer, 2012, Seiten 217–232. ISBN: 978-3-6423-5755-8. DOI: 10.1007/978-3-642-35755-8_16 (Zitiert auf S. 67).

[ME10] P. McDaniel und W. Enck. „Not So Great Expectations: Why Application Markets Haven't Failed Security". In: *IEEE Security and Privacy* 8.5 (Sep. 2010), Seiten 76–78. DOI: 10.1109/MSP.2010.159 (Zitiert auf S. 67).

[MFP06] G. Mühl, L. Fiege und P. Pietzuch. *Distributed Event-Based Systems*. Secaucus, NJ, USA: Springer, 2006. ISBN: 978-3-540-32653-3 (Zitiert auf S. 47).

[MHB10] S. Motiee, K. Hawkey und K. Beznosov. „Do Windows Users Follow the Principle of Least Privilege?: Investigating User Account Control Practices". In: *Proceedings of the Sixth Symposium on Usable Privacy and Security*. SOUPS '10. 2010, 1:1–1:13. DOI: 10.1145/1837110.1837112 (Zitiert auf S. 86).

[MHK15] E. L. Murnane, D. Huffaker und G. Kossinets. „Mobile Health Apps: Adoption, Adherence, and Abandonment". In: *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. UbiComp/ISWC '15 Adjunct. 2015, Seiten 261–264. DOI: 10.1145/2800835.2800943 (Zitiert auf S. 34, 227).

[Mic14] Microsoft. *Windows Phone 8.1 Security Overview*. Whitepaper. Microsoft Developer Network, Apr. 2014. URL: https://www.microsoft.com/en-us/download/details.aspx?id=42509 (Zitiert auf S. 54, 57).

[Mic16] Microsoft. *Microsoft Product Lifecycle*. Produktdatenblatt. Microsoft, Juli 2016. URL: https://support.microsoft.com/en-us/lifecycle/search?alpha=windows%20mobile%206.5 (Zitiert auf S. 50).

[Mil11] C. Miller. „Mobile Attacks and Defense". In: *IEEE Security and Privacy* 9.4 (Juli 2011), Seiten 68–70. DOI: 10.1109/MSP.2011.85 (Zitiert auf S. 52, 64).

[MK97] D. Mazières und M. F. Kaashoek. „Secure Applications Need Flexible Operating Systems". In: *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*. HOTOS '97. 1997, Seiten 56–61 (Zitiert auf S. 67).

[MMK+16] A. Mohan, H. Modi, K. Khandwala, K. Shreeja und Y. Shen. *Making Marshmallow's Permissions Sweet Again*. Technischer Bericht. Systems und Networking Group, University of California, San Diego, März 2016. URL: https://cseweb.ucsd.edu/classes/wi16/cse227-a/ (Zitiert auf S. 62).

[Mor10] S. Morrissey. *iOS Forensic Analysis for iPhone, iPad and iPod touch*. New York, NY, USA: Apress, 2010. ISBN: 978-1-4302-3342-8 (Zitiert auf S. 53).

[Mor11] T. Morris. „Trusted Platform Module“. In: *Encyclopedia of Cryptography and Security*. Herausgegeben von H. C. A. van Tilborg und S. Jajodia. Boston, MA, USA: Springer, 2011, Seiten 1332–1335. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_796 (Zitiert auf S. 57).

[MTDG11] A. Mylonas, B. Tsoumas, S. Dritsas und D. Gritzalis. „A Secure Smartphone Applications Roll-out Scheme“. In: *Trust, Privacy and Security in Digital Business: 8th International Conference, TrustBus 2011, Toulouse, France, August 29 - September 2, 2011. Proceedings*. Herausgegeben von S. Furnell, C. Lambrinoudakis und G. Pernul. Berlin, Heidelberg: Springer, 2011, Seiten 49–61. ISBN: 978-3-6422-2890-2. DOI: 10.1007/978-3-642-22890-2_5 (Zitiert auf S. 64).

[MVH12] K. W. Miller, J. Voas und G. F. Hurlburt. „BYOD: Security and Privacy Considerations“. In: *IT Professional* 14.5 (Sep. 2012), Seiten 53–55. DOI: 10.1109/MITP.2012.93 (Zitiert auf S. 232).

[MWC13] M. Ma, P. Wang und C.-H. Chu. „Data Management for Internet of Things: Challenges, Approaches and Opportunities“. In: *Proceedings of the 2013 IEEE World Cybermatics Congress (International Conference on Green Computing and Communications, International Conference on Cyber, Physical and Social Computing, and International Conference on the Internet of Things)*. GreenCom/CPSCom/iThings '13. 2013, Seiten 1144–1151. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.199 (Zitiert auf S. 234).

[New11] T. Newton. *Demystifying Shims—or—Using the App Compat Toolkit to make your old stuff work with your new stuff*. Entwicklerunterlagen. Microsoft, Juni 2011. URL: https://blogs.technet.microsoft.com/askperf/2011/06/17/demystifying-shims-or-using-the-app-compat-toolkit-to-make-your-old-stuff-work-with-your-new-stuff/ (Zitiert auf S. 89).

[Nic14] P. Nickinson. *New Google Play Store greatly simplifies permissions: Important permissions bundled into a primary group, with common permissions relegated to a secondary screen*. Pressemitteilung. Android Central, Juni 2014. URL: http://www.androidcentral.com/new-google-play-store-4820-greatly-simplifies-permissions (Zitiert auf S. 31).

[NKO+12] M. Nauman, S. Khan, A. T. Othman, S. u. Musa und N. U. Rehman. „POAuth: Privacy-aware Open Authorization for Native Apps on Smartphone Platforms“. In: *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*. ICUIMC ’12. 2012, 60:1–60:8. DOI: 10.1145/2184751.2184825 (Zitiert auf S. 217).

[NKW15] M. Naveed, S. Kamara und C. V. Wright. „Inference Attacks on Property-Preserving Encrypted Databases“. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. 2015, Seiten 644–655. DOI: 10.1145/2810103.2813651 (Zitiert auf S. 214).

[NKZ10] M. Nauman, S. Khan und X. Zhang. „Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints“. In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ASIACCS ’10. 2010, Seiten 328–332. DOI: 10.1145/1755688.1755732 (Zitiert auf S. 78, 166).

[NPD13] R. Neisse, A. Pretschner und V. Di Giacomo. „A Trustworthy Usage Control Enforcement Framework“. In: *International Journal of Mobile Computing and Multimedia Communications* 5.3 (Juli 2013), Seiten 34–49. DOI: 10.4018/jmcmc.2013070103 (Zitiert auf S. 92).

[OD16] L. Onwuzurike und E. De Cristofaro. „POSTER: Experimental Analysis of Popular Anonymous, Ephemeral, and End-to-End Encrypted Apps“. In: *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. WiSec ’16. 2016, Seiten 221–222. DOI: 10.1145/2939918.2942424 (Zitiert auf S. 220).

[OH12] J. O’Donoghue und J. Herbert. „Data Management Within mHealth Environments: Patient Sensors, Mobile Devices, and Databases“. In: *Journal of Data and Information Quality (JDIQ)* 4.1 (Okt. 2012), 5:1–5:20. DOI: 10.1145/2378016.2378021 (Zitiert auf S. 226).

[Ols12] G. Olsen. „Windows Phone 8 Security“. In: *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM ’12. 2012, Seiten 1–2. DOI: 10.1145/2381934.2381936 (Zitiert auf S. 58).

[Ols14] G. Olsen. *New Security Features for Windows Phone*. Build 2014 Vortrag. Microsoft, Apr. 2014. URL: https://channel9.msdn.com/Events/Build/2014/2-531 (Zitiert auf S. 58).

[OMEM09] M. Ongtang, S. McLaughlin, W. Enck und P. McDaniel. „Semantically Rich Application-Centric Security in Android“. In: *Proceedings of the 2009 Annual Computer Security Applications Conference*. ACSAC '09. 2009, Seiten 340–349. DOI: 10.1109/ACSAC.2009.39 (Zitiert auf S. 186, 238).

[OMEM12] M. Ongtang, S. McLaughlin, W. Enck und P. McDaniel. „Semantically rich application-centric security in Android“. In: *Security and Communication Networks* 5.6 (Juni 2012), Seiten 658–673. DOI: 10.1002/sec.360 (Zitiert auf S. 197).

[OTK+12] C. Orthacker, P. Teufl, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder und O. Prevenhueber. „Android Security Permissions – Can We Trust Them?“ In: *Security and Privacy in Mobile Information and Communication Systems: Third International ICST Conference, MobiSec 2011, Aalborg, Denmark, May 17-19, 2011, Revised Selected Papers*. Herausgegeben von R. Prasad, K. Farkas, A. U. Schmidt, A. Lioy, G. Russello und F. L. Luccio. Berlin, Heidelberg: Springer, 2012, Seiten 40–51. ISBN: 978-3-6423-0244-2. DOI: 10.1007/978-3-642-30244-2_4 (Zitiert auf S. 62).

[Pan13] A. Panos. „BYOD – Private Hardware in der Firma nutzen“. Diplomarbeit. Universität Stuttgart, Juli 2013 (Zitiert auf S. 125, 232).

[Par14] A. Park Eun. „Exploring the Multidimensionality of the Smartphone Divide: A New Aspect of the Digital Divide“. In: *Proceedings of the 36th Annual Pacific Telecommunications Conference*. PTC '14. 2014, Seiten 800–822 (Zitiert auf S. 29).

[Pay15] D. Payne. „MySugr“. In: *Nursing Standard* 29.33 (Apr. 2015), Seiten 31–31. DOI: 10.7748/ns.29.33.31.s34 (Zitiert auf S. 229).

[PHAB10] G. Portokalidis, P. Homburg, K. Anagnostakis und H. Bos. „Paranoid Android: Versatile Protection for Smartphones“. In: *Proceedings of the 26th Annual Computer Security Applications Conference*. ACSAC '10. 2010, Seiten 347–356. DOI: 10.1145/1920261.1920313 (Zitiert auf S. 87).

[Pil00] F. T. Piller. „Einführung: Informationsrevolution und industrielle Produktion“. In: *Mass Customization: Ein wettbewerbsstrategisches Konzept im Informationszeitalter*. Wiesbaden: Deutscher Universitätsverlag, 2000. Kapitel 1, Seiten 1–9. ISBN: 978-3-6630-8187-6. DOI: 10.1007/978-3-663-08187-6_1 (Zitiert auf S. 29).

[Pin15] M. Pinelis. *Apple's Smart Sensor Technologies*. Marktanalyse. MEMS Journal, Inc., Aug. 2015. URL: http://www.slideshare.net/MikePinelisPhD/apples-smart-sensor-technologies-market-research-report-sample (Zitiert auf S. 120).

[Poh04] H. Pohl. „Taxonomie und Modellbildung in der Informationssicherheit". In: *Datenschutz und Datensicherung* 28.11 (2004), Seiten 678–685 (Zitiert auf S. 40).

[PPH13] J.-H. Park, Y. B. Park und H. K. Ham. „Fragmentation Problem in Android". In: *Proceedings of the 2013 International Conference on Information Science and Applications*. ICISA '13. 2013, Seiten 1–2. DOI: 10.1109/ICISA.2013.6579465 (Zitiert auf S. 169).

[PS01] V. Prevelakis und D. Spinellis. „Sandboxing Applications". In: *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*. 2001, Seiten 119–126 (Zitiert auf S. 52).

[PS11] J. Posegga und D. Schreckling. „Next Generation Mobile Application Security". In: *IT-Sicherheit zwischen Regulierung und Innovation: Tagungsband zur zweiten EICT-Konferenz IT-Sicherheit*. Herausgegeben von U. Bub und K.-D. Wolfenstetter. Wiesbaden: Vieweg+Teubner Verlag, 2011. Kapitel 10, Seiten 181–199. ISBN: 978-3-8348-8256-1. DOI: 10.1007/978-3-8348-8256-1_10 (Zitiert auf S. 63, 67).

[PW14] A. Plaskett und N. Walker. *Windows Phone 8 Application Security*. Whitepaper. MWR InfoSecurity Labs, März 2014. URL: https://labs.mwrinfosecurity.com/publications/windows-phone-8-application-security-whitepaper-syscan-2014/ (Zitiert auf S. 54, 57).

[Qui16] Quick Heal. *Annual Threat Report 2016*. Whitepaper. Quick Heal Technologies Limited, 2016 (Zitiert auf S. 33).

[RA06] L. Rauchwerger und N. M. Amato. „SmartApps: Middle-ware for Adaptive Applications on Reconfigurable Platforms". In: *ACM SIGOPS Operating Systems Review* 40.2 (Apr. 2006), Seiten 73–82. DOI: 10.1145/1131322.1131338 (Zitiert auf S. 269).

[RCFZ11] G. Russello, B. Crispo, E. Fernandes und Y. Zhauniarovich. „YAASE: Yet Another Android Security Extension". In: *Proceedings of the 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Compu-*

ting (SocialCom). PASSAT '11. 2011, Seiten 1033–1040. DOI: 10.1109/PASSAT/SocialCom.2011.151 (Zitiert auf S. 79, 179).

[RCJ13] V. Rastogi, Y. Chen und X. Jiang. „DroidChameleon: Evaluating Android Anti-malware Against Transformation Attacks". In: *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. ASIA CCS '13. 2013, Seiten 329–334. DOI: 10.1145/2484313.2484355 (Zitiert auf S. 218).

[RGR16] T. Ringer, D. Grossman und F. Roesner. „AUDACIOUS: User-Driven Access Control with Unmodified Operating Systems". In: *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. 2016, Seiten 1–13 (Zitiert auf S. 84, 169, 170).

[RJV+11] N. Reddy, J. Jeon, J. A. Vaughan, T. Millstein und J. S. Foster. *Application-centric security policies on unmodified Android*. Technischer Bericht 110017. UCLA Computer Science Department, Juli 2011 (Zitiert auf S. 84).

[RMT01] W. Rosenblatt, S. Mooney und W. Trippe. *Digital Rights Management: Business and Technology*. New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN: 978-0-7645-4889-5 (Zitiert auf S. 92).

[Roo12] D. Rook. *Windows Phone 7 Security*. OWASP AppSec Vortrag. Realex Payments, Sep. 2012. URL: http://de.slideshare.net/securityninja/owasp-app-sec-ireland-windows-phone-7-security (Zitiert auf S. 56).

[RPP15] R. Rawassizadeh, B. A. Price und M. Petre. „Wearables: Has the Age of Smartwatches Finally Arrived?" In: *Communications of the ACM* 58.1 (Jan. 2015), Seiten 45–47. DOI: 10.1145/2629633 (Zitiert auf S. 226).

[RRBC13] J. Reardon, H. Ritzdorf, D. Basin und S. Capkun. „Secure Data Deletion from Persistent Media". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. 2013, Seiten 271–284. DOI: 10.1145/2508859.2516699 (Zitiert auf S. 199).

[RSA78] R. L. Rivest, A. Shamir und L. M. Adleman. „A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Communications of the ACM* 21.2 (Feb. 1978), Seiten 120–126. DOI: 10.1145/359340.359342 (Zitiert auf S. 64).

[RSC+11] C. Ran, D. Sun, G. Chang, L. Sun und X. Wang. „Surveying and Analyzing Security, Privacy and Trust Issues in Cloud Computing Environments“. In: *Procedia Engineering* 15 (2011), Seiten 2852–2856. DOI: 10.1016/j.proeng.2011.08.537 (Zitiert auf S. 38).

[RZCZ13] F. Rohrer, Y. Zhang, L. Chitkushev und T. Zlateva. „DR BACA: Dynamic Role Based Access Control for Android“. In: *Proceedings of the 29th Annual Computer Security Applications Conference*. ACSAC '13. 2013, Seiten 299–308. DOI: 10.1145/2523649.2523676 (Zitiert auf S. 217).

[Sad09] E. Sadun. *Das iPhone-Entwicklerbuch*. München: Addison-Wesley, 2009. ISBN: 978-3-8273-6211-7 (Zitiert auf S. 52).

[Sag12] I. Sager. *Before iPhone and Android Came Simon, the First Smartphone*. Pressemitteilung. Bloomberg, Juni 2012. URL: http://www.bloomberg.com/news/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone (Zitiert auf S. 28, 29).

[Sal14] D. Salsa. „Der PMP Gatekeeper“. Bachelorarbeit. Universität Stuttgart, Aug. 2014 (Zitiert auf S. 93, 143, 165).

[SB11] C. Stach und A. Brodt. „vHike – A Dynamic Ride-sharing Service for Smartphones“. In: *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management*. MDM '11. IEEE, Juni 2011, Seiten 333–336. DOI: 10.1109/MDM.2011.33 (Zitiert auf S. 221, 224).

[SBG+13] D. Sbırlea, M. G. Burke, S. Guarnieri, M. Pistoia und V. Sarkar. „Automatic Detection of Inter-application Permission Leaks in Android Applications“. In: *IBM Journal of Research and Development* 57.6 (Nov. 2013), 2:10–2:10. DOI: 10.1147/JRD.2013.2284403 (Zitiert auf S. 173).

[SBKN16] S. Steuer, A. Benabbas, N. Kasrin und D. Nicklas. „Challenges and Design Goals for an Architecture of a Privacy-preserving Smart City Lab“. In: *Datenbank-Spektrum* 16.2 (Juli 2016), Seiten 147–156. DOI: 10.1007/s13222-016-0223-8 (Zitiert auf S. 234).

[SBvO13] A. Skillen, D. Barrera und P. C. van Oorschot. „Deadbolt: Locking Down Android Disk Encryption“. In: *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. SPSM '13. 2013, Seiten 3–14. DOI: 10.1145/2516760.2516771 (Zitiert auf S. 212).

[SC13a] E. Schmidt und J. Cohen. *The New Digital Age: Reshaping the Future of People, Nations and Business*. New York City, NY, USA: Knopf Publishing Group, 2013. ISBN: 978-0-3079-5713-9 (Zitiert auf S. 27).

[SC13b] S. Smalley und R. Craig. „Security Enhanced (SE) Android: Bringing Flexible MAC to Android“. In: *Proceedings of the 20th Annual Network and Distributed System Security Symposium*. NDSS ’13. 2013, Seiten 1–18 (Zitiert auf S. 59).

[SCB15] S.-T. Sun, A. Cuadros und K. Beznosov. „Android Rooting: Methods, Detection, and Evasion“. In: *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM ’15. 2015, Seiten 3–14. DOI: 10.1145/2808117.2808126 (Zitiert auf S. 67).

[Sch04] F. B. Schneider. „Least Privilege and More“. In: *Computer Systems* 1.5 (2004), Seiten 55–59. DOI: 10.1109/MSECP.2003.1236236 (Zitiert auf S. 55).

[Sch10] J. Schiavo. „Code Signing for End-user Peace of Mind“. In: *Network Security* 2010.7 (Juli 2010), Seiten 11–13. DOI: 10.1016/S1353-4858(10)70093-3 (Zitiert auf S. 64).

[Sch13a] A. Schneider. „Kontextbezogene Informationsbereitstellung in der Fabrik“. Bachelorarbeit. Universität Stuttgart, Okt. 2013 (Zitiert auf S. 230).

[Sch13b] P. Scholz. „Integration der PMP in das Android OS“. Diplomarbeit. Universität Stuttgart, Apr. 2013 (Zitiert auf S. 76, 93, 140).

[Sch16] J. Schallaböck. *Sieben goldene Regeln des Datenschutzes*. Pressemitteilung. iRights.info, Jan. 2016. URL: http://www.bloomberg.com/news/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone (Zitiert auf S. 41, 42).

[SDB14] O. Serrano, L. Dandurand und S. Brown. „On the Design of a Cyber Security Data Sharing System“. In: *Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security*. WISCS ’14. 2014, Seiten 61–69. DOI: 10.1145/2663876.2663882 (Zitiert auf S. 220).

[SDC13] P. J. Sheehan, B. Davis und H. Chen. „Rewriting an Android App Using RetroSkeleton“. In: *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys ’13. 2013, Seiten 483–484. DOI: 10.1145/2462456.2465738 (Zitiert auf S. 84).

[SdLL10] R. A. Shaikh, B. J. d'Auriol, H. Lee und S. Lee. „Privacy and Trust Management Schemes of Wireless Sensor Networks: A Survey". In: *Handbook of Research on Developments and Trends in Wireless Sensor Networks: From Principle to Practice*. Herausgegeben von H. Jin und W. Jiang. Hershey, PA, USA: IGI Global, 2010. Kapitel 13, Seiten 290–309. ISBN: 978-1-6152-0701-5. DOI: 10.4018/978-1-61520-701-5.ch013 (Zitiert auf S. 37).

[SDM+17] C. Stach, F. Dürr, K. Mindermann, S. M. Palanisamy, M. A. Tariq, B. Mitschang und S. Wagner. „PATRON — Datenschutz in Datenstromverarbeitungssystemen". In: *Informatik 2017: Digitale Kulturen, Tagungsband der 47. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 25.9-29.9.2017, Chemnitz*. Band 275. LNI. GI, Sep. 2017, Seiten 1085–1096. ISBN: 978-3-88579-669-5. DOI: 10.18420/in2017_110 (Zitiert auf S. 279).

[SDX11] H. J. Smith, T. Dinev und H. Xu. „Information Privacy Research: An Interdisciplinary Review". In: *MIS Quarterly* 35.4 (Dez. 2011), Seiten 989–1016 (Zitiert auf S. 36).

[SFE+13] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck und J. Hoffmann. „Mobile-sandbox: Having a Deeper Look into Android Applications". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. SAC '13. 2013, Seiten 1808–1815. DOI: 10.1145/2480362.2480701 (Zitiert auf S. 87).

[SFK+10] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev und C. Glezer. „Google Android: A Comprehensive Security Assessment". In: *IEEE Security and Privacy* 8.2 (März 2010), Seiten 35–44. DOI: 10.1109/MSP.2010.2 (Zitiert auf S. 64, 72).

[Sha12] S. S. Shapiro. „The State and Evolution of Privacy by Design". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. 2012, Seiten 1053–1053. DOI: 10.1145/2382196.2382324 (Zitiert auf S. 262).

[Sie12] D. Siewiorek. „Generation Smartphone". In: *IEEE Spectrum* 49.9 (Sep. 2012), Seiten 54–58. DOI: 10.1109/MSPEC.2012.6281134 (Zitiert auf S. 225).

[SK10] B. Steeb und S. Kiesewetter. „vHike – a virtual hitchhiking system for mobile devices". Softwarepraktikum. Universität Stuttgart, Okt. 2010 (Zitiert auf S. 224).

[SKA+10] A. Sahami Shirazi, T. Kubitza, F. Alt, B. Pfleging und A. Schmidt. „WE-transport: A Context-based Ride Sharing Platform“. In: *Proceedings of the 12th ACM International Conference Adjunct Papers on Ubiquitous Computing - Adjunct*. UbiComp ’10 Adjunct. 2010, Seiten 425–426. DOI: 10.1145/1864431.1864469 (Zitiert auf S. 223).

[SLG+12] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru und I. Molloy. „Android Permissions: A Perspective Combining Risks and Benefits“. In: *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*. SACMAT ’12. 2012, Seiten 13–22. DOI: 10.1145/2295136.2295141 (Zitiert auf S. 86).

[SM13] C. Stach und B. Mitschang. „Privacy Management for Mobile Platforms – A Review of Concepts and Approaches“. In: *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management*. MDM ’13. IEEE, Juni 2013, Seiten 305–313. DOI: 10.1109/MDM.2013.45 (Zitiert auf S. 42, 43, 50, 76, 92, 180, 238).

[SM14] C. Stach und B. Mitschang. „Design and Implementation of the Privacy Management Platform“. In: *Proceedings of the 2014 IEEE 15th International Conference on Mobile Data Management*. MDM ’14. IEEE, Juli 2014, Seiten 69–72. DOI: 10.1109/MDM.2014.14 (Zitiert auf S. 44, 76, 93, 94, 135, 138, 147).

[SM15] C. Stach und B. Mitschang. „Der Secure Data Container (SDC) – Sicheres Datenmanagement für mobile Anwendungen“. In: *Datenbank-Spektrum* 15.2 (Juli 2015), Seiten 109–118. ISSN: 1618-2162. DOI: 10.1007/s13222-015-0189-y (Zitiert auf S. 42, 44, 111, 184, 193, 197, 238).

[SM16a] C. Stach und B. Mitschang. „The Secure Data Container: An Approach to Harmonize Data Sharing with Information Security“. In: *Proceedings of the 2016 IEEE 17th International Conference on Mobile Data Management*. MDM ’16. IEEE, Juni 2016, Seiten 292–297. DOI: 10.1109/MDM.2016.50 (Zitiert auf S. 42, 44, 184, 187, 205, 238, 242, 247).

[SM16b] D. Suarez und D. Mayer. „Faux Disk Encryption: Realities of Secure Storage on Mobile Devices“. In: *Proceedings of the International Conference on Mobile Software Engineering and Systems*. MOBILESoft ’16. 2016, Seiten 283–284. DOI: 10.1145/2897073.2897711 (Zitiert auf S. 220).

[Sma15] Smart-Data-Begleitforschung. *Smart Data – Smart Privacy? Impulse für eine interdisziplinär rechtlich-technische Evaluation*. Technischer Bericht. FZI Forschungszentrum Informatik, Nov. 2015 (Zitiert auf S. 34).

[SMAD16] A. Saracino, F. Martinelli, G. Alboreto und G. Dini. „Data-Sluice: Fine-grained traffic control for Android application“. In: *Proceedings of the 2016 IEEE Symposium on Computers and Communication*. ISCC ’16. 2016, Seiten 702–709. DOI: 10.1109/ISCC.2016.7543819 (Zitiert auf S. 79, 171).

[SML07] P. Stahlberg, G. Miklau und B. N. Levine. „Threats to Privacy in the Forensic Analysis of Database Systems“. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’07. 2007, Seiten 91–102. DOI: 10.1145/1247480.1247492 (Zitiert auf S. 198).

[Son16] Sony Developer World Mobile. *Unlock your boot loader*. Anleitung. Sony Mobile Communications Inc., Juli 2016. URL: http://developer.sonymobile.com/unlockbootloader/unlock-yourboot-loader/ (Zitiert auf S. 67).

[SPDS14] H. Shewale, S. Patil, V. Deshmukh und P. Singh. „Analysis of android vulnerabilities and modern exploitation techniques“. In: *ICTACT Journal on Communication Technology* 5.1 (März 2014), Seiten 863–867 (Zitiert auf S. 58).

[SPJW12] J. Snaddon, G. Petrokofsky, P. Jepson und K. J. Willis. „Biodiversity technologies: tools as change agents“. In: *Biology Letters* 9.1 (2012), Seiten 1–3. DOI: 10.1098/rsbl.2012.1029 (Zitiert auf S. 29).

[SRdlT+15] B. M. C. Silva, J. P. C. Rodrigues, I. de la Torre Díez, M. López-Coronado und K. Saleem. „Mobile-health: A Review of Current State in 2015“. In: *Journal of Biomedical Informatics* 56 (Aug. 2015), Seiten 265–272. DOI: 10.1016/j.jbi.2015.06.003 (Zitiert auf S. 225).

[SRGC15] S. Sicari, A. Rizzardi, L. Grieco und A. Coen-Porisini. „Security, Privacy and Trust in Internet of Things“. In: *Computer Networks* 76 (Jan. 2015), Seiten 146–164. DOI: 10.1016/j.comnet.2014.11.008 (Zitiert auf S. 38).

[SS12a] R. Schlöglhofer und J. Sametinger. „Secure and Usable Authentication on Mobile Devices“. In: *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*. MoMM ’12. 2012, Seiten 257–262. DOI: 10.1145/2428955.2429004 (Zitiert auf S. 61, 220).

[SS12b] C. Stach und L. F. M. Schlindwein. „Candy Castle — A Prototype for Pervasive Health Games“. In: *Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops*. PerCom ’12. IEEE, März 2012, Seiten 501–503. DOI: 10.1109/PerComW.2012.6197547 (Zitiert auf S. 221, 225).

[SS16] V. Singh und K. Sharma. „Smartphone Security: Review of Challenges and Solution“. In: *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*. ICTCS ’16. 2016, 8:1–8:3. DOI: 10.1145/2905055.2905214 (Zitiert auf S. 220).

[SS75] J. H. Saltzer und M. D. Schroeder. „The Protection of Information in Computer Systems“. In: *Proceedings of the IEEE* 63.9 (1975), Seiten 1278–1308. DOI: 10.1109/PROC.1975.9939 (Zitiert auf S. 55).

[SSF17] C. Stach, F. Steimle und A. C. Franco da Silva. „TIROL: The Extensible Interconnectivity Layer for mHealth Applications“. In: *Information and Software Technologies: 23nd International Conference, ICIST 2017, Druskininkai, Lithuania, October 12-14, 2017, Proceedings*. Herausgegeben von R. Damaševičius und V. Mikašytė. Band 756. Communications in Computer and Information Science. Cham: Springer, Okt. 2017, Seiten 190–202. ISBN: 978-3-31967-642-5. DOI: 10.1007/978-3-319-67642-5_16 (Zitiert auf S. 227).

[SSP+12] J. Sorber, M. Shin, R. Peterson, C. Cornelius, S. Mare, A. Prasad, Z. Marois, E. Smithayer und D. Kotz. „An Amulet for Trustworthy Wearable mHealth“. In: *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. HotMobile ’12. 2012, 7:1–7:6. DOI: 10.1145/2162081.2162092 (Zitiert auf S. 227).

[Sta09] C. Stach. „Mobile ortsbasierte Browserspiele“. Diplomarbeit. Universität Stuttgart : Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Germany, Jan. 2009 (Zitiert auf S. 221, 222).

[Sta10a] C. Stach. „Gamework – A Customizable Framework for Pervasive Games“. In: *Proceedings of the 2010 IADIS International Conference Game and Entertainment Technologies*. MCCSIS ’10. IADIS, Juli 2010, Seiten 45–52. ISBN: 978-972-8939-18-2 (Zitiert auf S. 266, 268).

[Sta10b] C. Stach. „Gamework – A customizable framework for pervasive games“. In: *Proceedings of the 7th International Conference on Pervasive Services*. ICPS '10. ACM, Juli 2010, Seiten 168–173. ISBN: 978-1-4503-0249-4 (Zitiert auf S. 266, 269, 270).

[Sta11] C. Stach. „Saving time, money and the environment – vHike a dynamic ride-sharing service for mobile devices“. In: *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops*. PerCom '11. **IEEE PerCom 2011 Best WiP Poster Award**. IEEE, März 2011, Seiten 352–355. DOI: 10.1109/PERCOMW.2011.5766904 (Zitiert auf S. 221, 224).

[Sta12] C. Stach. „Gamework – A Framework Approach for Customizable Pervasive Applications“. In: *International Journal of Computer Information Systems and Industrial Management Applications* 4 (Jan. 2012), Seiten 66–75. ISSN: 2150-7988 (Zitiert auf S. 266, 269).

[Sta13a] C. Stach. „How to Assure Privacy on Android Phones and Devices?“ In: *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management*. MDM '13. IEEE, Juni 2013, Seiten 350–352. DOI: 10.1109/MDM.2013.54 (Zitiert auf S. 92).

[Sta13b] C. Stach. „Wie funktioniert Datenschutz auf Mobilplattformen?“ In: *Informatik 2013: Informatik angepasst an Mensch, Organisation und Umwelt, Tagungsband der 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 16.9-20.9.2013, Koblenz*. Band 220. LNI. GI, Sep. 2013, Seiten 2072–2086. ISBN: 978-3-88579-614-5 (Zitiert auf S. 43, 50, 71, 72, 76, 83, 92, 93, 150, 180).

[Sta15a] C. Stach. „How to Deal with Third Party Apps in a Privacy System — The PMP Gatekeeper“. In: *Proceedings of the 2015 IEEE 16th International Conference on Mobile Data Management*. MDM '15. IEEE, Juni 2015, Seiten 167–172. DOI: 10.1109/MDM.2015.17 (Zitiert auf S. 44, 79, 93, 137, 145, 156, 266).

[Sta15b] C. Stach. *Privacy_Management_Platform: Privacy Management Platform for Android* 6.0. GitHub Project. Okt. 2015. DOI: 10.5281/zenodo.32005. URL: https://github.com/stachch/Privacy_Management_Platform (Zitiert auf S. 93, 135, 148).

[Sta16] C. Stach. „Secure Candy Castle — A Prototype for Privacy-Aware mHealth Apps“. In: *Proceedings of the 2016 IEEE 17th International Conference on Mobile Data Management*. MDM ’16. IEEE, Juni 2016, Seiten 361–364. DOI: 10.1109/MDM.2016.64 (Zitiert auf S. 184, 221, 227, 228, 357).

[Stat13] Statista. *Number of mobile app downloads worldwide from 2009 to 2017 (in millions)*. Pressemitteilung. Statista GmbH, Sep. 2013. URL: http://www.statista.com/statistics/266488/forecast-of-mobile-app-downloads/ (Zitiert auf S. 30).

[Stat16] Statista. *Number of apps available in leading app stores as of June 2016*. Pressemitteilung. Statista GmbH, Juni 2016. URL: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (Zitiert auf S. 30).

[STCT15] Sufatrio, D. J. J. Tan, T.-W. Chua und V. L. L. Thing. „Securing Android: A Survey, Taxonomy, and Challenges“. In: *ACM Computing Surveys (CSUR)* 47.4 (Juli 2015), 58:1–58:45. DOI: 10.1145/2733306 (Zitiert auf S. 81).

[Sto15] Stollmann. *Terminal I/O Profile*. Datenblatt. Stollmann Entwicklungs- und Vertriebs-GmbH, 2015 (Zitiert auf S. 226).

[Sva13] V. Svajcer. *Not Just for PCs Anymore: The Rise of Mobile Malware*. Whitepaper. SophosLabs, 2013 (Zitiert auf S. 33).

[Sva14] V. Svajcer. *Sophos Mobile Security Threat Report*. Whitepaper. SophosLabs, 2014 (Zitiert auf S. 31, 33).

[SVEG09] E. Shmueli, R. Vaisenberg, Y. Elovici und C. Glezer. „Database Encryption: An Overview of Contemporary Challenges and Design Considerations“. In: *ACM SIGMOD Record* 38.3 (Nov. 2009), Seiten 29–34. DOI: 10.1145/1815933.1815940 (Zitiert auf S. 194).

[SW05] A. Sahai und B. Waters. „Fuzzy Identity-Based Encryption“. In: *Advances in Cryptology – EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*. Herausgegeben von R. Cramer. Berlin, Heidelberg: Springer, 2005, Seiten 457–473. ISBN: 978-3-5403-2055-5. DOI: 10.1007/11426639_27 (Zitiert auf S. 211).

[SWM+15] F. Steimle, M. Wieland, B. Mitschang, S. Wagner und F. Leymann. „Design and Implementation Issues of a Secure Cloud-Based Health Data Management System“. In: *Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing*. SummerSOC '15. 2015, Seiten 68–82 (Zitiert auf S. 228).

[Tee15] C. Teegarden. *Runtime Permissions: Best Practices and How to Gracefully Handle Permission Removal*. Pressemitteilung. CapTech Ventures, Inc., Sep. 2015. URL: https://www.captechconsulting.com/blogs/runtime-permissions-best-practices-and-how-to-gracefully-handle-permission-removal (Zitiert auf S. 265).

[Tej13] A. Tejaswi. *Android malicious, data theft on the rise*. Pressemitteilung. Deccan Chronicle, Aug. 2013. URL: http://archives.deccanchronicle.com/130811/news-businesstech/article/android-malicious-data-theft-rise (Zitiert auf S. 183).

[TGHI13] A. G. F. Teacher, D. J. Griffiths, D. J. Hodgson und R. Inger. „Smartphones in ecology and evolution: a guide for the app-rehensive“. In: *Ecology and Evolution* 3.16 (Dez. 2013), Seiten 5268–5278. DOI: 10.1002/ece3.888 (Zitiert auf S. 29).

[Tho12] G. Thomson. „BYOD: enabling the chaos“. In: *Network Security* 2012.2 (Feb. 2012), Seiten 5–8. DOI: 10.1016/S1353-4858(12)70013-2 (Zitiert auf S. 232).

[Tra12] K. W. Tracy. „Mobile Application Development Experiences on Apple's iOS and Android OS“. In: *IEEE Potentials* 31.4 (Juli 2012), Seiten 30–34. DOI: 10.1109/MPOT.2011.2182571 (Zitiert auf S. 30).

[Tre16] Trend Micro. *Mobile Security für Android-Smartphones und -Tablets*. Produktdatenblatt. Trend Micro Incorporated, Juli 2016. URL: http://www.trendmicro.de/produkte/mobile-security-for-android/ (Zitiert auf S. 218).

[TUP16] B. Tank, H. Upadhyay und H. Patel. „A Survey on IoT Privacy Issues and Mitigation Techniques“. In: *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*. ICTCS '16. 2016, 2:1–2:4. DOI: 10.1145/2905055.2905057 (Zitiert auf S. 234).

[UC16] R. Unuchek und V. Chebyshev. *Mobile malware evolution 2015*. Sicherheitsreport. AO Kaspersky Lab, Feb. 2016. URL: https://securelist.com/analysis/kaspersky-security-bulletin/73839/mobile-malware-evolution-2015/ (Zitiert auf S. 53).

[VA15] U. Vignesh und S. Asha. „Modifying Security Policies Towards BYOD“. In: *Procedia Computer Science* 50.2015 (Mai 2015), Seiten 511–516. DOI: 10.1016/j.procs.2015.04.023 (Zitiert auf S. 232).

[Ved11] A. Vedder. „Privacy 3.0“. In: *Innovating Government: Normative, Policy and Technological Dimensions of Modern Government*. Herausgegeben von S. van der Hof und M. M. Groothuis. The Hague: T. M. C. Asser Press, 2011. Kapitel 3, Seiten 17–28. ISBN: 978-9-0670-4731-9. DOI: 10.1007/978-90-6704-731-9_2 (Zitiert auf S. 37).

[Vil13] N. M. Villegas Machado. „Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems“. Dissertation. University of Victoria, Feb. 2013 (Zitiert auf S. 270).

[VJB+12] M. Vetter, J. Jarosch, T. Berberich, H. D. Huynh, T. Kuhn, A. Makarov, A. B. Nguyen, F. Schäler, P. Strobel und A. Wassiljew. „Aufbau einer Privacy Management Platform für kontextsensitive mobile Apps“. Studienprojekt. Universität Stuttgart, Juni 2012 (Zitiert auf S. 93, 135, 224, 266).

[Wai15] T. Waizenegger. „SDOS: Secure Deletion in the Swift Object Store“. In: *Poster Presentation at the 9th Symposium and Summer School On Service-Oriented Computing*. SummerSOC ’15. 2015, Seiten 1–1 (Zitiert auf S. 199).

[Wai17] T. Waizenegger. „Secure Cryptographic Deletion in the Swift Object Store“. In: *Tagungsband der 15. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web*. BTW ’17. 2017, Seiten 627–630 (Zitiert auf S. 199).

[War98] P. D. Warner. „ACL for Windows“. In: *The CPA Journal* 68.11 (Nov. 1998), Seiten 40–44 (Zitiert auf S. 56).

[WAX15] A. Whitmore, A. Agarwal und L. Xu. „The Internet of Things – A Survey of Topics and Trends“. In: *Information Systems Frontiers* 17.2 (Apr. 2015), Seiten 261–274. DOI: 10.1007/s10796-014-9489-2 (Zitiert auf S. 233).

[WCS+02] C. Wright, C. Cowan, S. Smalley, J. Morris und G. Kroah-Hartman. „Linux Security Modules: General Security Support for the Linux Kernel“. In: *Proceedings of the 11th USENIX Security Symposium*. USENIX Security '02. 2002, Seiten 17–31 (Zitiert auf S. 80).

[WDSH12] M. Wolf, K. Dirtheuer, S. Sagl und G. Herrmann. „Mobil - interaktiv - sozial: Der digitale Mensch von morgen - “always in touch”?“ In: *Smart Mobile Apps: Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Herausgegeben von S. Verclas und C. Linnhoff-Popien. Berlin, Heidelberg: Springer, 2012. Kapitel 35, Seiten 531–544. ISBN: 978-3-6422-2259-7. DOI: 10.1007/978-3-642-22259-7_35 (Zitiert auf S. 27).

[Wei91] M. Weiser. „The computer for the 21st century“. In: *Scientific American* 265.3 (1991), Seiten 94–104. DOI: 10.1038/scientificamerican0991-94 (Zitiert auf S. 28).

[WGZ+13] L. Wu, M. Grace, Y. Zhou, C. Wu und X. Jiang. „The Impact of Vendor Customizations on Android Security“. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. 2013, Seiten 623–634. DOI: 10.1145/2508859.2516728 (Zitiert auf S. 81).

[WHD+13] T. Werthmann, R. Hund, L. Davi, A.-R. Sadeghi und T. Holz. „PSiOS: Bring Your Own Privacy & Security to iOS Devices“. In: *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. ASIA CCS '13. 2013, Seiten 13–24. DOI: 10.1145/2484313.2484316 (Zitiert auf S. 76).

[Whi15] L. Whitney. *Microsoft cooks up way to run Windows 10 on Android devices*. Pressemitteilung. CNET, März 2015. URL: http://www.cnet.com/news/microsoft-cooks-up-way-to-run-windows-10-on-android-devices/ (Zitiert auf S. 54).

[WKS08] C. Wright, D. Kleiman und S. Sundhar R.S. „Overwriting Hard Drive Data: The Great Wiping Controversy“. In: *Information Systems Security: 4th International Conference, ICISS 2008, Hyderabad, India, December 16-20, 2008. Proceedings*. Herausgegeben von R. Sekar und A. K. Pujari. Berlin, Heidelberg: Springer, 2008, Seiten 243–257. ISBN: 978-3-5408-9862-7. DOI: 10.1007/978-3-540-89862-7_21 (Zitiert auf S. 198).

[WLL+13] T. Wang, K. Lu, L. Lu, S. Chung und W. Lee. „Jekyll on iOS: When Benign Apps Become Evil“. In: *Proceedings of the 22nd USENIX Conference on Security*. SEC '13. 2013, Seiten 559–572 (Zitiert auf S. 53).

[WM12a] M. E. Whitman und H. J. Mattord. *Principles of Information Security*. 4th edition. Boston, MA, USA: Course Technology, Cengage Learning, 2012. ISBN: 978-1-1111-3821-9 (Zitiert auf S. 35).

[WM12b] P. Wildt und R. Meister. „Das Smartphone als sichere Burg“. In: *Smart Mobile Apps: Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Herausgegeben von S. Verclas und C. Linnhoff-Popien. Berlin, Heidelberg: Springer, 2012. Kapitel 14, Seiten 209–223. ISBN: 978-3-6422-2259-7. DOI: 10.1007/978-3-642-22259-7_14 (Zitiert auf S. 58).

[WM13] A. Whitechapel und S. McKenna. *Windows Phone 8 Development Internals*. Developer Reference. Sebastopol, CA, USA: O'Reilly Media, Inc., 2013. ISBN: 978-0-7356-7623-7 (Zitiert auf S. 54, 56, 57).

[WMM13] C. Walker-Osborn, S. Mann und V. Mann. „to BYOD or . . . not to BYOD“. In: *ITNOW* 55.1 (2013), Seiten 38–39. DOI: 10.1093/itnow/bws142 (Zitiert auf S. 232).

[WMVF03] R. Watson, W. Morrison, C. Vance und B. Feldman. „The TrustedBSD MAC framework: Extensible kernel access control for FreeBSD 5.0“. In: *Proceedings of the 2003 USENIX Annual Technical Conference, FREENIX Track*. USENIX ATC '03. 2003, Seiten 285–296 (Zitiert auf S. 80).

[WP00] G. Wolf und A. Pfitzmann. „Charakteristika von Schutzzielen und Konsequenzen für Benutzungsschnittstellen“. In: *Informatik-Spektrum* 23.3 (2000), Seiten 173–191. DOI: 10.1007/s002870000101 (Zitiert auf S. 41).

[WP14] H. Walther und S. Pereira. *Exchange ActiveSync Considerations When Using Windows Phone 7 Clients*. Entwicklerunterlagen. Microsoft, Juni 2014. URL: https://social.technet.microsoft.com/wiki/contents/articles/1769.exchange-activesync-considerations-when-using-windows-phone-7-clients.aspx (Zitiert auf S. 57).

[WR13] R. Wilkins und B. Richardson. *UEFI Secure Boot in Modern Computer Security Solutions*. Whitepaper. UEFI Forum, Sep. 2013. URL: http://staging.uefi.org/learning_center/papers (Zitiert auf S. 54).

[WR16] M. Wagner und O. Raabe. „7 Irrtümer zum Datenschutz im Kontext von Smart Data“. In: *Datenbank-Spektrum* 16.2 (2016), Seiten 173–178. DOI: 10.1007/s13222-016-0218-5 (Zitiert auf S. 41, 42).

[WRR09] D. Weerasinghe, M. Rajarajan und V. Rakocevic. „Device Data Protection in Mobile Healthcare Applications“. In: *Electronic Healthcare: First International Conference, eHealth 2008, London, UK, September 8-9, 2008. Revised Selected Papers*. Herausgegeben von D. Weerasinghe. Berlin, Heidelberg: Springer, 2009, Seiten 82–89. ISBN: 978-3-6420-0413-1. DOI: 10.1007/978-3-642-00413-1_10 (Zitiert auf S. 227).

[WvdM16] V. Woods und R. van der Meulen. *Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016*. Pressemitteilung. Gartner, Inc., Mai 2016. URL: http://www.gartner.com/newsroom/id/3323017 (Zitiert auf S. 29, 49).

[WZSI14] X. Wang, J. Zhang, E. M. Schooler und M. Ion. „Performance evaluation of Attribute-Based Encryption: Toward data privacy in the IoT“. In: *Proceedings of the 2014 IEEE International Conference on Communications*. ICC '14. 2014, Seiten 725–730. DOI: 10.1109/ICC.2014.6883405 (Zitiert auf S. 211).

[XBL+15] L. Xing, X. Bai, T. Li, X. Wang, K. Chen, X. Liao, S.-M. Hu und X. Han. „Cracking App Isolation on Apple: Unauthorized Cross-App Resource Access on MAC OS X and iOS“. In: *Proceeding of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CSS '15. 2015, Seiten 31–43. DOI: 10.1145/2810103.2813609 (Zitiert auf S. 53).

[XSA12] R. Xu, H. Saıdi und R. Anderson. „Aurasium: Practical Policy Enforcement for Android Applications“. In: *Proceedings of the 21st USENIX Conference on Security Symposium*. Security '12. 2012, Seiten 539–552 (Zitiert auf S. 84, 170).

[YH13] Z. Yan und S. Holtmanns. „Trust Modeling and Management: From Social Trust to Digital Trust“. In: *Examining the Concepts, Issues, and Implications of Internet Trolling*. Herausgegeben von J. Bishop. Hershey, PA, USA: IGI Global, 2013. Kapitel 18, Seiten 279–303. ISBN: 978-1-4666-2803-8. DOI: 10.4018/978-1-4666-2803-8 (Zitiert auf S. 38).

[YN11] D.-H. You und B.-N. Noh. „Android Platform Based Linux Kernel Rootkit“. In: *Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software*. MALWARE '11. 2011, Seiten 79–87. DOI: 10.1109/MALWARE.2011.6112330 (Zitiert auf S. 67).

[YWS+14] X. Yu, Z. Wang, K. Sun, W. T. Zhu, N. Gao und J. Jing. „Remotely Wiping Sensitive Data on Stolen Smartphones“. In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS '14. 2014, Seiten 537–542. DOI: 10.1145/2590296.2590318 (Zitiert auf S. 215).

[YY12] Z. Yang und M. Yang. „LeakMiner: Detect Information Leakage on Android with Static Taint Analysis“. In: *Proceedings of the 2012 Third World Congress on Software Engineering*. WCSE '12. 2012, Seiten 101–104. DOI: 10.1109/WCSE.2012.26 (Zitiert auf S. 184).

[YZW+14] K. Yang, J. Zhuge, Y. Wang, L. Zhou und H. Duan. „IntentFuzzer: Detecting Capability Leaks of Android Applications“. In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS '14. 2014, Seiten 531–536. DOI: 10.1145/2590296.2590316 (Zitiert auf S. 243).

[ZAH+14] R. Zhou, Z. Ai, J. Hu, Q. Liu, Q. Zhou, X. Wang, H. Jiang und K.-C. Li. „Data Integrity Checking for iSCSI with Dm-verity“. In: *Advanced Technologies, Embedded and Multimedia for Human-centric Computing: HumanCom and EMC 2013*. Herausgegeben von Y.-M. Huang, H.-C. Chao, D.-J. Deng und J. J. Park. Dordrecht: Springer, 2014, Seiten 691–697. ISBN: 978-9-4007-7262-5. DOI: 10.1007/978-94-007-7262-5_79 (Zitiert auf S. 58).

[ZGM12] L. Zhang, D. Gupta und P. Mohapatra. „How Expensive are Free Smartphone Apps?“ In: *ACM SIGMOBILE Mobile Computing and Communications Review* 16.3 (Juli 2012), Seiten 21–32. DOI: 10.1145/2412096.2412100 (Zitiert auf S. 262).

[ZN06] P. Zheng und L. M. Ni. „Spotlight: The Rise of the Smart Phone“. In: *IEEE Distributed Systems Online* 7.3 (März 2006), Seiten 1–14. DOI: 10.1109/MDSO.2006.22 (Zitiert auf S. 221).

[ZSL14] M. Zheng, M. Sun und J. C. Lui. „DroidRay: A Security Evaluation System for Customized Android Firmwares“. In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS '14. 2014, Seiten 471–482. DOI: 10.1145/2590296.2590313 (Zitiert auf S. 88).

[ZWL+16] D. Zhang, R. Wang, Z. Lin, D. Guo und X. Cao. „IacDroid: Preventing Inter-App Communication capability leaks in Android“. In: *Proceedings of the 2016 IEEE Symposium on Computers and Communication*. ISCC ’16. 2016, Seiten 443–449. DOI: 10.1109/ISCC.2016.7543779 (Zitiert auf S. 79, 173).

[ZWZJ12] Y. Zhou, Z. Wang, W. Zhou und X. Jiang. „Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets“. In: *Proceedings of the 19th Network and Distributed System Security Symposium*. NDSS ’12. 2012, Seiten 1–13 (Zitiert auf S. 68).

[ZXMX13] Y. Zhongyang, Z. Xin, B. Mao und L. Xie. „DroidAlarm: An All-sided Static Analysis Tool for Android Privilege-escalation Malware“. In: *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. ASIA CCS ’13. 2013, Seiten 353–358. DOI: 10.1145/2484313.2484359 (Zitiert auf S. 86).

Alle URLs wurden zuletzt am 30. 11. 2017 geprüft.

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

ANHANG A

DATENSCHEMATA

In diesem Kapitel werden die Schemadefinitionen der beiden Extensible Markup Language (XML)-Dokumenttypen *Resource Group Information Set (RGIS)* (siehe Quelltext A.1) sowie *Application Information Set (AIS)* (siehe Quelltext A.2) präsentiert. Beide Schemata sind in XML Schema Definition (XSD)-Notation angegeben.

Eine Beschreibung des *RGISs* findet sich in Abschnitt 4.1.1 und das *AIS* ist Gegenstand von Abschnitt 4.1.3.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="resourceGroupInformationSet">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="resourceGroupInformation">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="name" type="translate" maxOccurs="unbounded"/>
      <xs:element name="description" type="translate"
       ↪ maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute type="xs:string" name="identifier" use="required"/>
     <xs:attribute type="xs:string" name="icon" use="required"/>
     <xs:attribute type="xs:string" name="className" use="required"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="privacySettings">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="privacySetting" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="name" type="translate" maxOccurs="unbounded"/>
         <xs:element name="description" type="translate"
          ↪ maxOccurs="unbounded"/>
         <xs:element name="changeDescription" type="translate"
          ↪ maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute type="xs:string" name="identifier" use="required"/>
        <xs:attribute type="xs:string" name="validValueDescription"
         ↪ use="required"/>
        <xs:attribute type="xs:string" name="requestable" use="required"/>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:complexType name="translate">
```

```
40  <xs:simpleContent>
41   <xs:extension base="xs:string">
42    <xs:attribute type="xs:string" name="lang" use="required"/>
43   </xs:extension>
44  </xs:simpleContent>
45 </xs:complexType>
46
47 </xs:schema>
```

Quelltext A.1: XML-Schema für das *RGIS*

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="appInformationSet">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="appInformation">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="name" type="translate" maxOccurs="unbounded"/>
      <xs:element name="description" type="translate"
       ↪ maxOccurs="unbounded"/>
      <xs:element name="serviceFeatures">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="serviceFeature" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="name" type="translate"
             ↪ maxOccurs="unbounded"/>
            <xs:element name="description" type="translate"
             ↪ maxOccurs="unbounded"/>
            <xs:element name="changeDescription" type="translate"
             ↪ maxOccurs="unbounded"/>
            <xs:element name="requiredResourceGroup">
             <xs:complexType>
              <xs:sequence>
               <xs:element name="requiredPrivacySetting"
                ↪ maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                 <xs:simpleContent>
                  <xs:extension base="xs:string">
                   <xs:attribute type="xs:string" name="identifier"
                    ↪ use="required"/>
                  </xs:extension>
                 </xs:simpleContent>
                </xs:complexType>
               </xs:element>
              </xs:sequence>
              <xs:attribute type="xs:string" name="identifier"
               ↪ use="required"/>
              <xs:attribute type="xs:byte" name="minRevision"
               ↪ use="required"/>
```

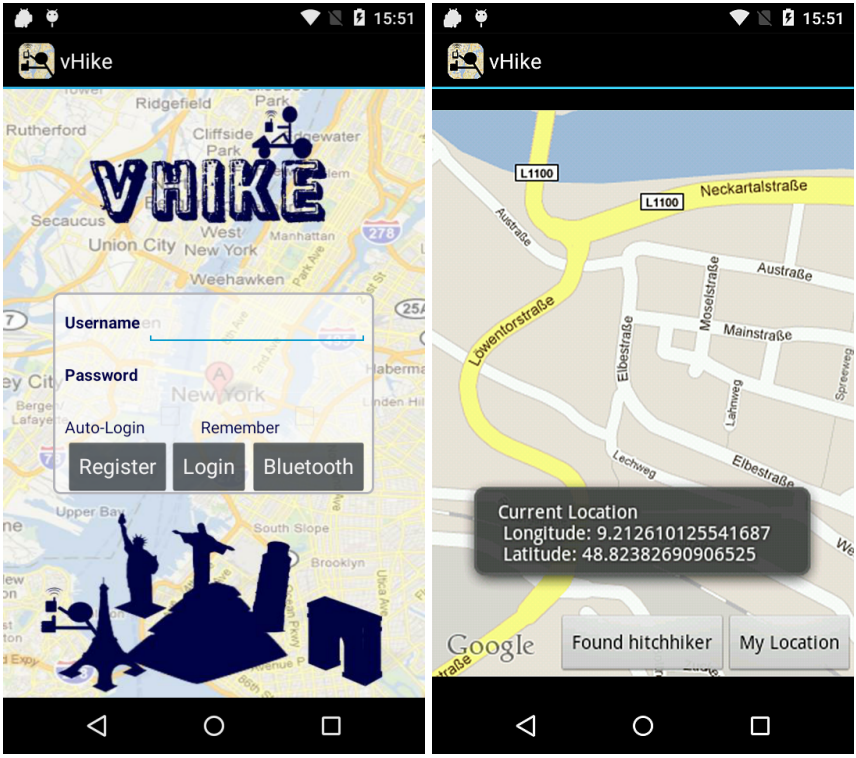
```
36              </xs:complexType>
37             </xs:element>
38            </xs:sequence>
39            <xs:attribute type="xs:string" name="identifier"
               ↪ use="required"/>
40           </xs:complexType>
41          </xs:element>
42         </xs:sequence>
43        </xs:complexType>
44       </xs:element>
45      </xs:sequence>
46     </xs:complexType>
47    </xs:element>
48   </xs:sequence>
49  </xs:complexType>
50 </xs:element>
51 <xs:complexType name="translate">
52  <xs:simpleContent>
53   <xs:extension base="xs:string">
54    <xs:attribute type="xs:string" name="lang" use="required"/>
55   </xs:extension>
56  </xs:simpleContent>
57 </xs:complexType>
58
59 </xs:schema>
```

Quelltext A.2: XML-Schema für das *AIS*

Anhang B

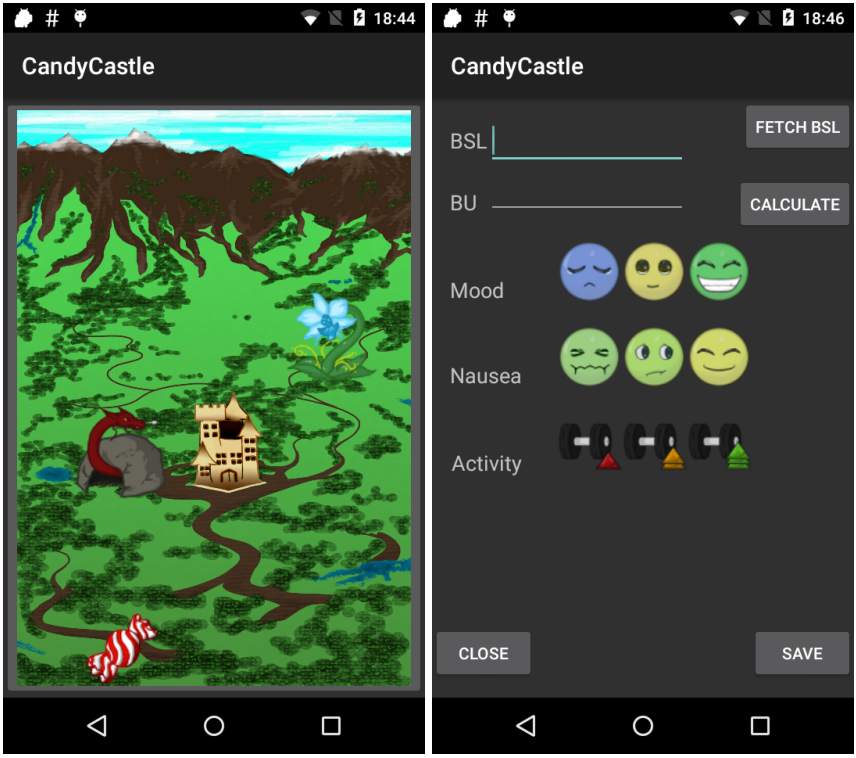
Bildschirmabbildungen der App-Prototypen

In diesem Kapitel werden Bildschirmabbildungen der App-Prototypen, die im Rahmen der vorliegenden Arbeit entwickelt wurden, präsentiert. Dabei handelt es sich um die ortsbasierte Anwendung *vHike*, ein Ad-hoc-Mitfahrdienst (siehe Abbildung B.1), die *mHealth*-Anwendung *Secure Candy Castle (SCC)*, ein Gesundheitsspiel für an Diabetes erkrankte Kinder (siehe Abbildung B.2), sowie die Industrie-4.0-Anwendung *Mobile Manufacturing Dashboard (MMD)*, eine interaktive und kontextsensitive Instrumententafel für den Fertigungsbereich (siehe Abbildung B.3).
Für weitere Informationen über diese drei Apps sei an dieser Stelle auf die jeweiligen Kapitel verwiesen; *vHike* wird in Abschnitt 6.1 näher betrachtet, *SCC* in Abschnitt 6.2 und das *MMD* in Abschnitt 6.3.



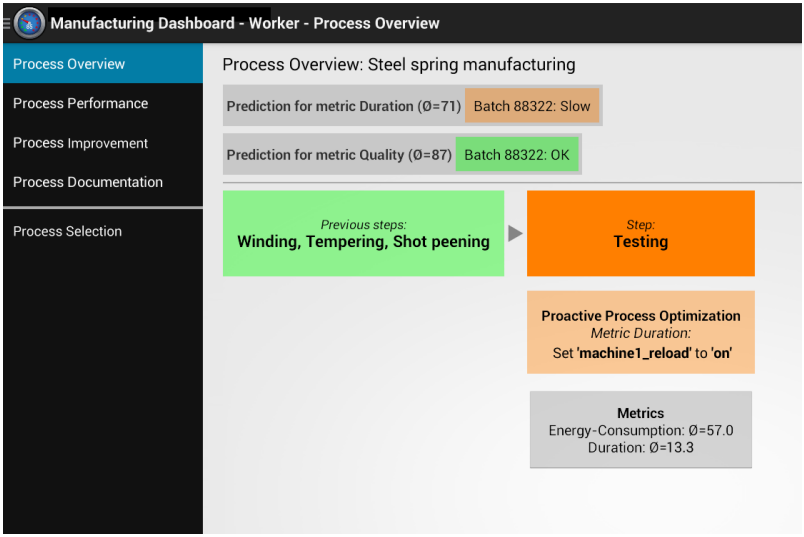
(a) Hauptmenü von *vHike* (b) Kartenansicht von *vHike*

Abbildung B.1: Bildschirmabbildungen von *vHike*

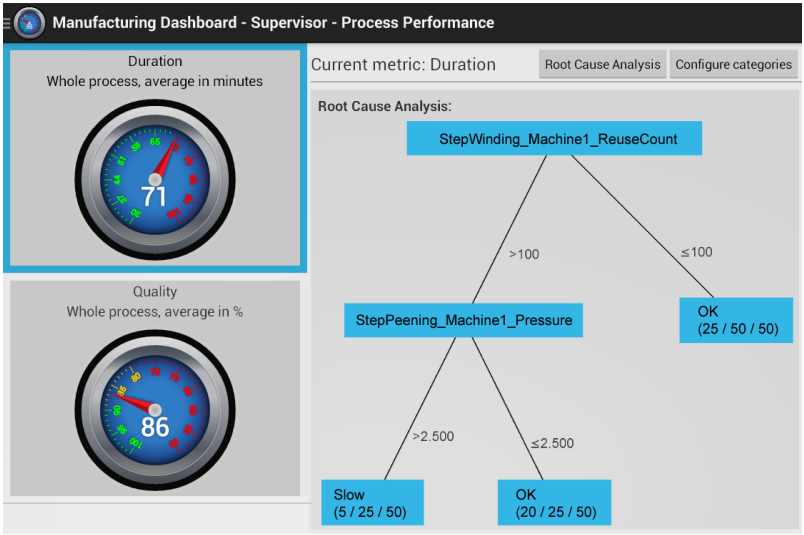


(a) Spielfeld von *SCC* (b) Gesundheitsdateneingabemaske

Abbildung B.2: Bildschirmabbildungen von *SCC* [Sta16]



(a) *MMD* für Arbeiter



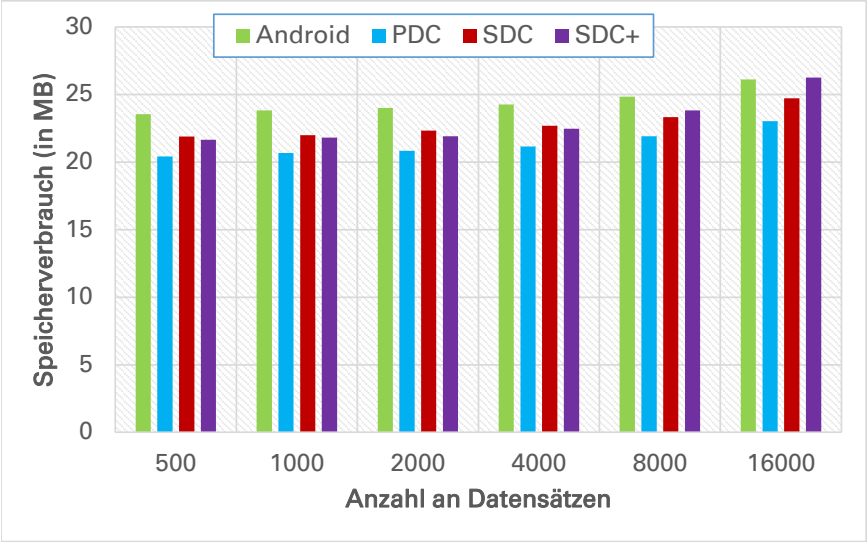
(b) *MMD* für Industriemeister

Abbildung B.3: Bildschirmabbildungen des *MMDs* [GS14]

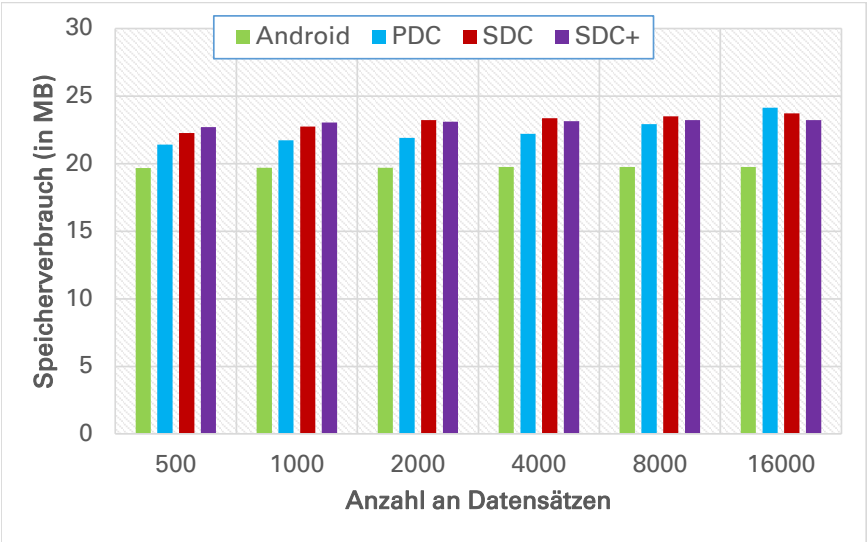
Anhang C

Zusätzliche Messergebnisse

In diesem Kapitel werden weitere Messergebnisse, die im Rahmen der technischen Evaluation (siehe Abschnitt 7.2) erfasst wurden, grafisch aufbereitet. Hierbei handelt es sich um den maximalen Speicherverbrauch des Schreib- und Lese-Benchmarks (siehe Abbildung C.1 für die *Proportional Set Size (PSS)* und Abbildung C.2 für die *Private Dirty Pages*), die Größe der verwendeten Datenbank in Abhängigkeit zu den geschriebenen Nutzdaten (siehe Abbildung C.3) sowie die App-Größe der beiden Benchmarks respektive die kumulierte Größe bestehend aus App und *Resource* (siehe Abbildung C.4).
Eine tabellarische Auflistung der Messwerte ist in Tabelle 7.3 und in Tabelle 7.4 (für den maximalen Speicherverbrauch sowie die Datenbankgröße) respektive in Tabelle 7.5 (für die App-Größe) gegeben. Die Ergebnisse werden in Abschnitt 7.2 und Abschnitt 7.3 diskutiert.

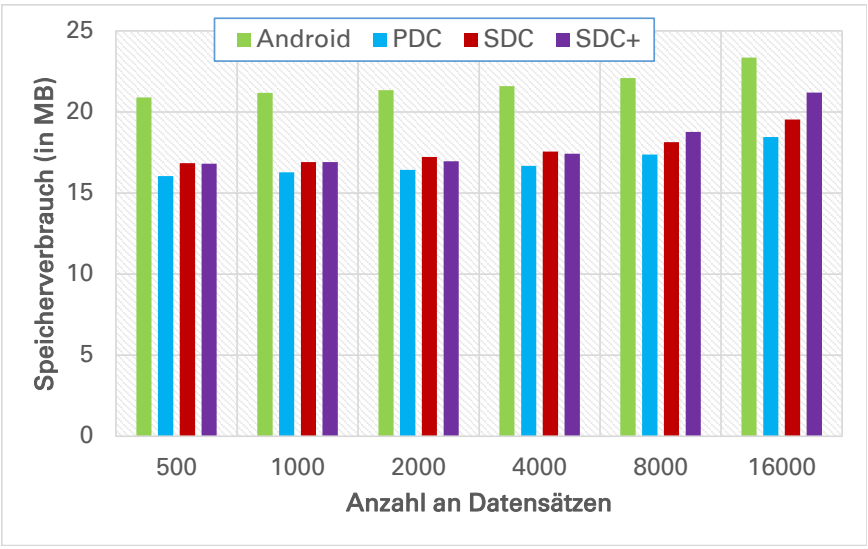


(a) Schreib-Benchmark

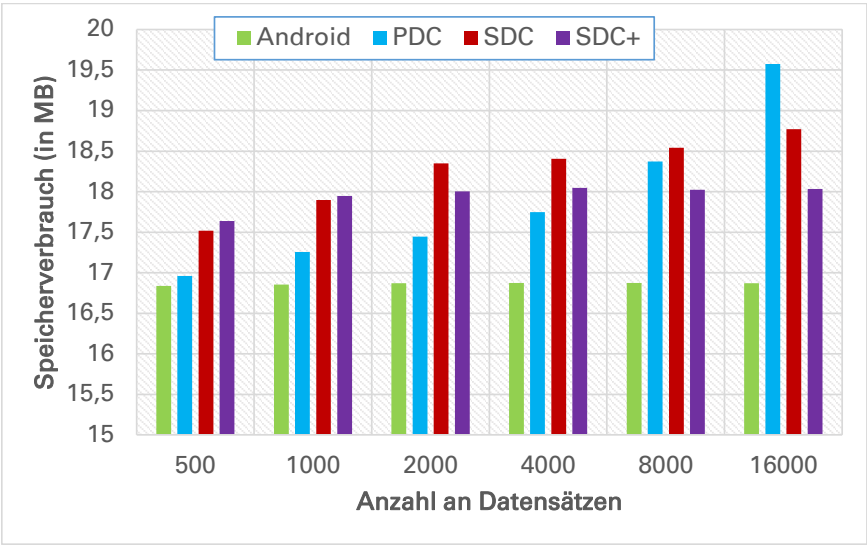


(b) Lese-Benchmark

Abbildung C.1: Messergebnisse: Speicherverbrauch (*PSS*)



(a) Schreib-Benchmark



(b) Lese-Benchmark

Abbildung C.2: Messergebnisse: Speicherverbrauch (*Private Dirty*)

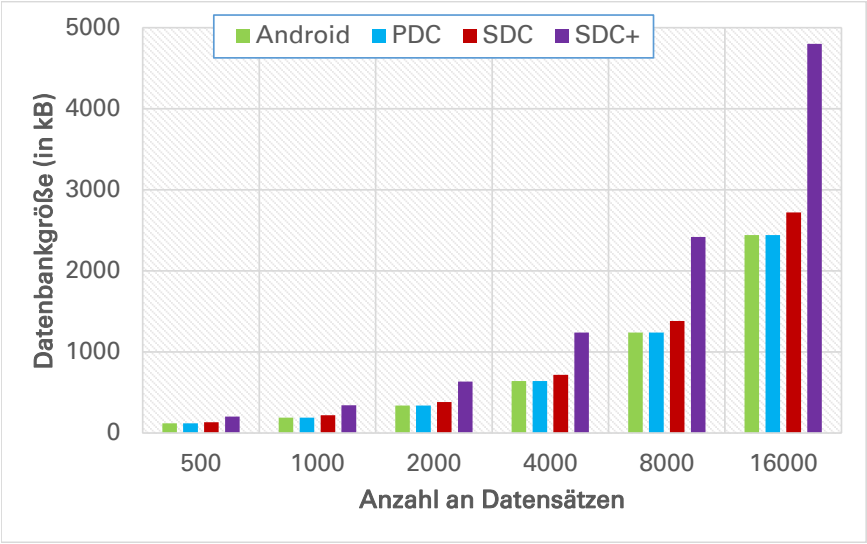


Abbildung C.3: Messergebnisse: Datenbankgröße

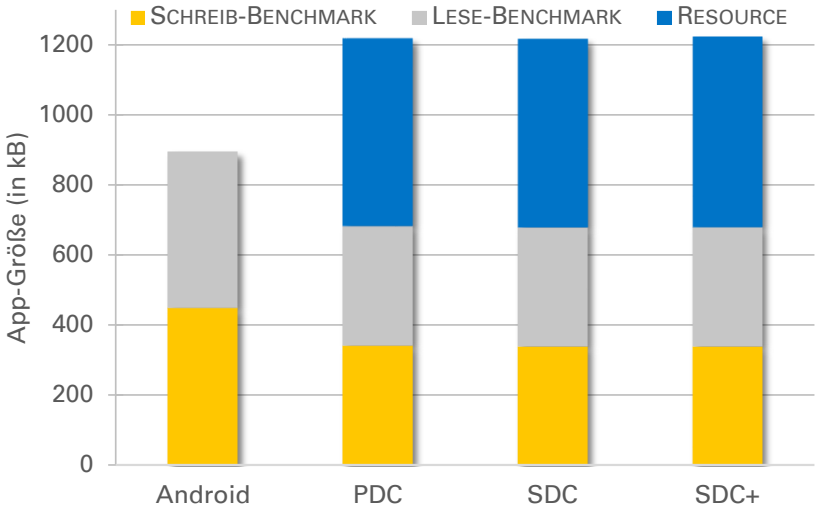


Abbildung C.4: Messergebnisse: App-Größe

ANHANG D

BILDSCHIRMABBILDUNGEN DER WERKZEUGUNTERSTÜTZUNG

In diesem Kapitel werden Bildschirmabbildungen der im Rahmen der vorliegenden Arbeit entwickelten Werkzeugunterstützung für die *Privacy Management Platform (PMP)* präsentiert. Dabei handelt es sich um den *Editor for Information Sets (EIS)*, einer *Eclipse*-Erweiterung zur Erstellung des *AISs* und des *RGISs*. Die Maske zur Spezifikation der *Service Features* im *AIS* ist in Abbildung D.1 dargestellt und Abbildung D.2 zeigt die Maske, mit dem die *Privacy Settings* im *RGIS* erzeugt werden können.
Für weitere Informationen zum *EIS* sei an dieser Stelle auf Abschnitt 8.1 verwiesen.

Abbildung D.1: Bildschirmabbildung der *AIS*-Maske

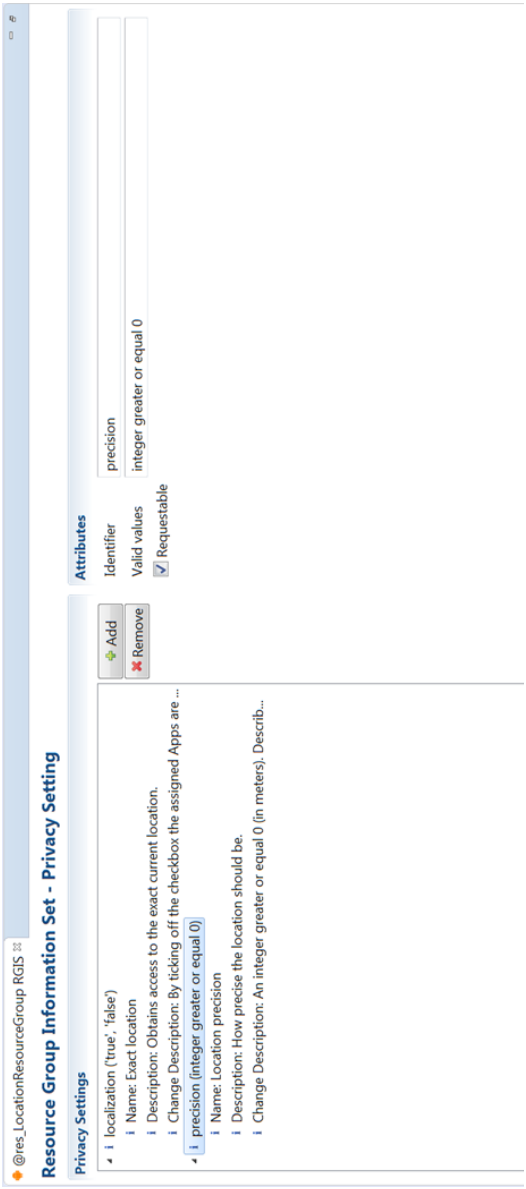


Abbildung D.2: Bildschirmabbildung der *RGIS*-Maske

Curriculum Vitae

Christoph Stach	
Geburtsdatum:	04. Juli 1981
Geburtsort:	Backnang, Deutschland
Staatsangehörigkeit:	deutsch
08/1988 – 06/1992	Grundschule Lippoldsweiler
08/1992 – 06/2001	Bildungszentrum Weissach im Tal **Abschluss:** *allgemeine Hochschulreife*
09/2001 – 06/2002	Ersatzdienst, Gemeinde Auenwald
10/2002 – 01/2009	Informatikstudium an der Universität Stuttgart, Deutschland **Abschluss:** *Diplom-Informatiker*
07/2008 – 01/2009	Diplomarbeit an der Universität Stuttgart, Deutschland **Titel:** *„Mobile ortsbasierte Browserspiele“*
04/2009 – heute	Wissenschaftlicher Angestellter am Institut für Parallele und Verteilte Systeme, Universität Stuttgart, Deutschland

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorgelegte Dissertation mit dem Titel „Konzepte für Datensicherheit und Datenschutz in mobilen Anwendungen“ selbständig und ohne unzulässige fremde Hilfe angefertigt und verfasst habe, dass alle Hilfsmittel und sonstigen Hilfen angegeben und dass alle Stellen, die ich wörtlich oder dem Sinne nach aus anderen Veröffentlichungen entnommen habe, kenntlich gemacht worden sind.
Ich versichere außerdem, dass die vorgelegte Arbeit in dieser oder in einer ähnlichen Fassung noch nicht zu einem früheren Zeitpunkt an der Universität Stuttgart oder einer anderen in- oder ausländischen Hochschule als Dissertation eingereicht wurde. Diesem Promotionsverfahren sind auch keine endgültig gescheiterten Promotionsverfahren vorausgegangen.

Ort, Datum

Christoph Stach