Dirk Westhoff

Mobile Security

Schwachstellen verstehen und Angriffsszenarien nachvollziehen

EBOOK INSIDE

Springer Vieweg

Mobile Security

Dirk Westhoff

Mobile Security

Schwachstellen verstehen und Angriffsszenarien nachvollziehen

Springer Vieweg

Dirk Westhoff
Offenburg, Baden-Württemberg, Deutschland

ISBN 978-3-662-60854-8 ISBN 978-3-662-60855-5 (eBook)
https://doi.org/10.1007/978-3-662-60855-5

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Planung/Lektorat: Martin Börger
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer-Verlag GmbH, DE und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Heidelberger Platz 3, 14197 Berlin, Germany

Man sieht nur, was man weiß.
Johann Wolfgang von Goethe

Für Charlotte, Johanna und Jutta

Vorwort

Es sind schon eine ganze Reihe Bücher zum Thema mobile und drahtlose Sicherheit verfasst worden [1–5], sodass man sich mit Recht fragen kann, inwieweit ein weiteres Buch zu diesem Thema sinnvoll erscheint. Die Anregung hierzu liefert der Begriff *„Selected Areas of…"*. Bei der Erstellung der Mastervorlesung „Mobile Security" im Studiengang Enterprise- and IT-Security (ENITS) an der Hochschule Offenburg hat sich gezeigt, dass dieses Feld (bis zum Jahre 2019) so mannigfaltig angewachsen ist, dass es sich in dem Format einer ‚Zwei-plus-zwei'-Veranstaltung innerhalb eines Semesters nicht annähernd abdecken lässt. Allerdings sollte ein Fachbuch aus meiner Sicht auch nicht deutlich umfangreicher sein, um die Leserschaft nicht schon aufgrund der schieren Seitenzahl abzuschrecken. So möchte ich mich, auch wenn der Themenbereich Mobile Security sicherlich einige Anknüpfungspunkte an das Thema *Social-Engineering*-Angriffe bietet, darauf beschränken, hierzu auf [6] hinzuweisen. Ähnliches gilt für den technischen Datenschutz, oder *privacy-by-design.* Auch wenn dieses Thema zum Ende des Buches angeschnitten wird so möchte ich für ein tieferes Studium auf [7] verweisen.

Die rasante Entwicklung digitaler Technologien wird offenkundig, wenn man sich zwei inzwischen berühmte Zitate vergegenwärtigt: So behauptete Thomas John Watson, Vorstandsvorsitzender bei IBM, im Jahre 1943: *„Ich glaube, dass es wohl weltweit einen Markt für vielleicht fünf Computer gibt."*. Ein wenig weitsichtiger formuliert hat es Mark Weiser, Chef-Entwickler bei Rank Xerox, Palo Alto Research Center, USA, mit seinem 1991 geäußerten Begriff des *„Ubiquitous Computing"*. Dieser ein halbes Jahrhundert nach Watson geformte Begriff des allgegenwärtigen Rechnens kumulierte eine ganze Reihe technologischer Entwicklungen. Er prognostizierte quasi implizit die Säulen der modernen IT: die Möglichkeit der Massenfertigung durch die Chipindustrie, massiv erweiterte Adressräume, deutlich kleinere Formfaktoren der Geräte, tragbare digitale sowie eingebettete digitale Geräte, die mehrheitlich drahtlos kommunizieren. Nun, die Prognose Mark Weisers hat sich in schon fast verstörender Weise bewahrheitet und die Zukunft dürfte aller Voraussicht nach wohl noch *„ubiquitärer"* werden. Wer dabei einordnen möchte, welche Autonomieverschiebungen durch Informations- und Kommunikationstechnologie derartige Entwicklungen der Gesellschaft abverlangen, der sei beispielsweise auf [8] verwiesen.

Das Anliegen dieses Buches ist es nunmehr, durch eine gezielte Auswahl einzelner Schwerpunkte dem Leser einen soliden Einblick in Fragestellungen rund um die IT-Sicherheit mobiler drahtloser digitaler Consumer-Geräte zu geben und ihm die dahinterliegenden Architekturen respektive deren Angreifbarkeit zu vermitteln. Dabei sollen uns nicht nur Fragen der Verwundbarkeit einzelner Technikkomponenten interessieren, sondern oftmals auch Aspekte der Privatheit der Nutzer solcher Systeme.

Offenburg, Deutschland Prof. Dr. Dirk Westhoff
Dezember 2019

Literatur

1. Buttyan, L., Hubaux, J.P.: Security and Cooperation in Wireless Networks – Thwarting Malicious and Selfish Behavior in the Age of Ubiquitous Computing. Cambridge University Press, Cambridge (2007)
2. Chen, Y., Xu, W., Trappe, W., Zheng, Y.: Securing Emerging Wireless Systems – Lower-Layer Approaches. Springer, New York (2009)
3. Osterhage, W.: Sicher & Mobil: Sicherheit in der drahtlosen Kommunikation. Springer, Heidelberg (2010)
4. Spreitzenbarth, M.: Mobile Hacking. dpunkt, Heidelberg (2017)
5. Stajano, F.: Security for Ubiquitous Computing. Wiley, Chichester (2002)
6. Drechsler, D. (Hrsg.): Schutz vor Social Engineering – Angriffspunkte und Abwehrmöglichkeiten in digitalwirtschaftlichen Ökosystemen. Schmidt, Berlin (2019)
7. Petrlic, R., Sorge, C.: Datenschutz – Einführung in technischen Datenschutz, Datenschutzrecht und angewandte Kryptographie. Springer Vieweg, Wiesbaden (2017)
8. Westhoff, D.: Gedanken zu Autonomieverschiebungen durch Informations- und Kommunikationstechnologie. In: Breyer-Mayländer, T. (Hrsg.) Das Streben nach Autonomie – Reflexionen zum Digitalen Wandel, S. 67–79. Nomos, Baden-Baden (2018)

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACK	Acknowledgment
ACM	Association for Computing Machinery
AES	Advanced Encryption Standard
AMI	Advanced Metering Infrastructure
BDSG	Bundesdatenschutzgesetz
BSI	Bundesamt für Sicherheit in der Informationstechnik
BSSID	Basic Service Set Identifier
CBC	Cipher block chaining
CCM	Counter with CBC-MAC
CCMP	Counter-Mode/CBC-MAC Protocol
CPS	Cipher-physisches System
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CSS	Cascading Style Sheets
CTR	Counter mode
DAC	Discretionary Access Control
DES	Data Encryption Standard
DLP	Discrete Logarithm Problem
DoS	Denial of Service
DSGVO	Datenschutz-Grundverordnung
EAPOL	Extensible Authentication Protocol over Local Area Network
ENITS	Enterprise and IT-Security
ETSI	Europäisches Institut für Telekommunikationsnorme
FFT	Fast Fourier-Transformation
GCHQ	Government Communications Headquarters
GCMP	Galois/Counter Mode
GPS	Global positioning system

HKPK	HTTP Public Key Pinning
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
ICV	Integrity Check Value
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
IPC	Inter Process Communication
IV	Initialisierungsvektor
LBS	Location based service
LFSR	linear feedback shift register
MAC	Mandatory Access Control
MAC	medium access control
MIC	message integrity code
MitM	Man in the middle
NFC	near field communication
NIC	network Interface card
NIDS	network intrusion detection system
NIPS	network intrusion prevention
NIST	National Institute of Standards an Technologies
Nonce	number used only once
OMA	Open mobile alliance
OWASP	Open Web Application Security Project
PAN	Personal Area Network
PHY	Physical Layer
PIN	Persönliche Identifikationsnummer
PKI	Public Key Infratructure
PMK	Pairwise Master Key
PRNG	Pseudorandom Number Generator
RFC	Request for Comments
RPC	Remote Procedure Call
RSSI	Received Singal Strength Indication
SCADA	Supervisory Control and Data Acquisition
SE	Secure Element
SSID	Service Set Identifier
TEE	Trusted Execution Environment
TKG	Telekommunikationsgesetz
TKIP	Temporal Key Integrity Protocol
TLS	Transport Layer Security
ToFU	Trust-on-first-use
TRNG	True Random Number Generator

WLAN	Wireless Local Area Network
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
USB	Universal Serial Bus

Abbildungsverzeichnis

Einführung

1

Es ist wohlbekannt, dass mobile eingebettete Geräte schon jetzt in ihrer Anzahl einen signifikanten und in ihrer wirtschaftlichen Bedeutung höchst relevanten Anteil an der heutigen Netzlandschaft darstellen. Beispielsweise spielen CPUs für Laptops und PCs im Mikroprozessormarkt zahlenmäßig mit etwa 1 % nur eine geringe Rolle, während der Anteil eingebetteter Prozessoren bei weit über 90 % liegt. Dies gilt insbesondere für Deutschland, da hier neben der stark anwachsenden Anzahl mobiler digitaler Geräte für den Consumer-Bereich (Smartphone, Tablet, Smartwatch) einer vergleichsweise schwach ausgeprägten PC-Industrie auch Schlüsselindustrien wie Maschinenbau, Automobil oder Medizintechnik gegenüberstehen, bei denen eingebettete Geräte von zentraler Bedeutung sind. Während der letzten Dekade wurden diese Geräte immer mehr vernetzt, wobei funkbasierte Anbindungen zunehmend dominieren. Drahtlose mobile Endgeräte sind allerdings aufgrund ihrer Ressourcenbeschränktheit und dem per se offenen Übertragungsmedium Luft prominente Angriffsziele. Fälle aus der jüngeren Vergangenheit wie der Stuxnet-Virus [1] oder Angriffe gegen Fahrzeuge über die GSM-Schnittstelle [2] haben zudem das erhebliche und im Fall Stuxnet nicht nur hypothetische Schadenspotenzial von Angriffen auf eingebettete Geräte aufgezeigt. In der Zukunft werden eingebettete und oftmals mobile Geräte und Systeme, die besonders eng mit der physikalischen Umgebung verbunden sind, eine sehr wichtige Rolle spielen. Unter diese sogenannten Cyber-physische Systeme (CPS) fallen viele gerade für die deutsche Wirtschaft wichtige Industrien, z. B. Sensornetze in der Automatisierung, Überwachung und Steuerung von Stromnetzen mittels ‚Supervisory Control and Data Acquisition' (SCADA)-Systemen, Smart Metering, Sensoren in der Medizintechnik und zahlreiche weitere als „Internet of Things (IoT)" bzw. Industrie 4.0 bezeichnete Anwendungen.

Neben der Sicherheit werden in zunehmendem Maße Lösungen zu entwickeln sein, unter anderem aufgrund von einer steigenden Zahl mobiler Geräte beispielsweise durch fahrerlose Transportsysteme, aber auch durch Einbindung von Smartphone und Tablet

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_1

in diverse digitale Geschäftsprozesse. Drahtlose Nachrichtenübertragungssysteme stellen hierfür geeignete Technologien der Nachrichtenübertragung dar. Die Verwendung eines gemeinsam genutzten Übertragungsmediums macht drahtlose Nachrichtenübertragungssysteme allerdings verwundbar, sowohl im Hinblick auf die Zuverlässigkeit und Verfügbarkeit der Nachrichtenverbindung, z. B. bei Interferenz, als auch wegen ihrer Verwundbarkeit gegenüber Angriffen auf dem gemeinsam genutzten Übertragungsmedium. Die Systeme bedürfen daher einer verlässlichen Nachrichtenverbindung – sowohl im Sinne der Verfügbarkeit als auch im Sinne der Echtheit, Vertraulichkeit und Einmaligkeit der Daten – über die eingesetzten Kommunikationssysteme. Insbesondere aufgrund langer Innovationszyklen in Industrieanlagen ist die Bereitstellung von verlässlichen Nachrichtenverbindungen auf Grundlage verfügbarer Technologien der drahtlosen Kommunikation daher von großer Bedeutung. Eine zusätzliche Angriffsfläche ist der Tatsache geschuldet, dass die mobilen Betriebssysteme und die auf ihnen laufenden Anwendungen heutiger mobiler Consumer-Geräte eine Reihe Einfallstore bereitstellen. Gleichzeitig offenbart sich, dass das Ausspähen sensibler Daten zur Wirtschaftsspionage deutlich auf dem Vormarsch ist [3].

1.1 Was nicht Gegenstand dieses Buches ist

Wenn man eine Themenauswahl trifft, dann bedeutet es immer auch, dass einige Themen zwangsläufig nicht aufgenommen werden können. Daher skizzieren wir nun einzelne Themen, die wir im Rahmen dieses Buches nicht weiterverfolgen, an denen sich der Leser jedoch die Weitläufigkeit des Feldes bewusst machen kann:

Funktionale Sicherheit Bevor man sich an die Absicherung von Systemen zum Schutze gegen verschiedenste Arten von Angreifern macht, ist es unerlässlich, die funktionale Sicherheit eines Systems, also die Sicherstellung seiner korrekten Funktion, zu gewährleisten. So sind uns allen die Meldungen zu explodierten Akkus im Smartphone-Modell Samsung Galaxy 7 noch gut in Erinnerung. Sie verdeutlichen, auf welch vielfältige Aspekte Ingenieure und Produktentwickler eingehen müssen, um ein System ganzheitlich funktionsfähig, ausfallsicher, verfügbar, robust bzw. fehlertolerant zu gestalten. So behandelt beispielsweise die Norm EN/IEC 61508 die funktionale Sicherheit sicherheitsbezogener elektrischer, elektronischer und/oder programmierbarer Systeme.

Biometrische Authentifizierungsverfahren Die Nutzerauthentifizierung auf einem Smartphone mittels biometrischen Finger-Scanning ist noch viel zu einfach zu umgehen. Es reicht schon das Foto eines Fingers mit einer hochauflösenden digitalen Kamera, das dem Smartphone vorgehalten wird. Dies gilt auch für aktuelle Smartphones wie das im März 2019 erschienene Samsung Galaxy S10 mit Ultraschall-Sensoren, welche im Display verbaut sind. Vielleicht ist das ja ein Grund für die konsequente Raute-Stellung der Hände unserer Kanzlerin. Hat man es mit einem Smartphone-Nutzer zu tun, der derart

bedacht auf seine Handstellung ist und seinen Zeigefinger auf gar keinen Fall ablichten lassen möchte, so kann dessen biometrische Authentifizierung dennoch mittels Fingerabdrücken an Gegenständen wie beispielsweise Gläsern unterlaufen werden. Ähnliches gilt für biometrische Authentifizierungsverfahren mittels Gesichtserkennung: Auch hier ist es noch viel zu einfach, die Gesichtserkennungs-Software auszutricksen, indem man dem Smartphone mit der Gesichtserkennungs-Software einfach ein weiteres Smartphone mit dem Gesicht der Person auf dem Display gegenhält. Und schließlich: Biometrische Authentifizierungskennungen, die einmal genutzt als digitaler Repräsentant in die Hände von Betrügern fallen, können nicht wie das gute alte Passwort nach einiger Zeit geändert werden, sondern können von nun an fortwährend zum Identitätsdiebstahl verwendet werden.

Verwendung von USB-Kabeln Die Verwendung eines Universal-Serial-Bus-(USB-)Kabels ermöglicht immer eine zweiseitige Kommunikation. Die Sicherheitsproblematik ist damit inhärent. Wenn Sie also wieder einmal im Drogeriemarkt Ihre Urlaubsbilder ausdrucken, dann sollte Ihnen bewusst sein, dass hierbei auch Malware auf Ihr Smartphone aufgespielt werden könnte.

Stauvorhersage Seit einiger Zeit profitieren wir von den Stauvorhersagen durch Google Maps. Dabei tragen wir Smartphone-Nutzer aktiv zu diesem Location Based Service (LBS) bei, indem das Unternehmen Google Nutzer-Tracking durchführt. Bei iPhones betrifft dies all diejenigen Geräte, die Google Maps aktiviert haben, und bei Android-Geräten diejenigen, deren Lokationsdienste aktiviert sind. Diese Geräte senden ‚anonymisierte' Daten an Google. Auf Basis dieser Daten kann das Unternehmen Google nun auf die Gesamtzahl der Fahrzeuge auf einem Streckenabschnitt schließen und ermitteln, wie schnell diese sich fortbewegen, und zwar auf jeder Straße und zu jeder Zeit.

IMSI-Catcher und stille SMS Mit OpenBTS, OsmonconBB oder OpenLTE stehen neben Produkten wie IMSI-Catcher PKI 1640 eine Reihe von Lösungen für einen IMSI-Catcher-Angriff bereit. Somit kann die International Mobile Subscriber Identity (IMSI) des Mobiltelefons sowie dessen gegenwärtiger Standort in Erfahrung gebracht werden. Ebenso dient das Absetzen von stiller SMS der Erzeugung eines Ortungsimpulses für mobile Geräte. Dabei ist der Einsatz von stiller SMS als Ermittlungswerkzeug für Behörden insbesondere hinsichtlich der Tatsache zu diskutieren, dass hierbei die aktive Erzeugung eines Ortungsimpulses erforderlich ist. Dies ist mit dem klassischen Einsatz eines IMSI-Catchers nicht der Fall. Denn letztere Technologie agiert rein passiv. Das Eingehen einer stillen SMS zum Erhalt von Verbindungsinformationen durch den Mobilfunkanbieter wird der Nutzer des mobilen Gerätes nicht gewahr. Mit HushSMS war lange Zeit eine App im Google Store verfügbar mit der ebenfalls SMS-Class-0-Ortungsimpulse versendet werden konnten.

Pager In Deutschland und Frankreich werden dedizierte Infrastrukturen auf Basis von NP2M-Techologie (Narrowband Point-to-Multipoint) betrieben, um im Falle von Katastrophen unabhängig von bekannten Netzbetreibern zu sein. Es wird eine Schmalbandtechnologie verwendet mit dem vorrangigen Ziel, versorgungssicher zu sein. Dabei geht es darum, eine große Anzahl von Empfängern in kürzester Zeit zu erreichen. Durch eine Standardisierung des ETSI (das Europäische Institut für Telekommunikationsnormen) aus dem Jahre 2013 ist es ideal für einen landes- und europaweiten Alamierungs- und Informationsdienst. Allerdings hat sich herausgestellt, dass Pager wie e*message auch sehr leicht manipulierbar sind. Dies ist gerade als Kommunikationswahl in Katastrophenfällen nicht hinnehmbar.

Prozessoren Mit Spectre1, Spectre2, Meltdown und Spectre Next Generation sind seit 2017 eine Reihe von sehr ernstzunehmenden Seitenkanalangriffen auf Chiparchitekturen bekannt geworden, die auch in mobilen Geräten verbaut sind. Dabei kann man sich einen Seitenkanal als eine geteilte Ressource vorstellen, die ursprünglich nicht zum Austausch von Informationen konzipiert worden ist, jedoch findig hinsichtlich dieser Eigenschaft zweckentfremdet wurde. Damit ist jedes digitale Gerät, welches solche Chips (Intel, Cortex-A75) hardwareseitig verbaut hat, anfällig gegenüber derartigen Angriffen. Solche Seitenkanalangriffe die im konkreten Fall auf Sprungvorhersage und auf einer Ausführung von Instruktionsabfolgen in einer anderen Reihenfolge als vorgesehen *(out-of-order execution)* basieren, ermöglichen einen unautorisierten Zugriff auf die Speicherbereiche fremder Prozesse. Und dies auch dann, wenn dem aufrufenden Prozess für diesen Prozess keine Zugriffsrechte vorliegen. Da die anfälligen Prozessoren auch in Smartphones und Tablets verwendet werden, sind diese Schwachstellen auch für mobile IT-Systeme von sehr großer Relevanz.

1.2 Was die Themen dieses Buches sind

Nachdem wir nun einen Eindruck gewinnen konnten, was dieses Buch inhaltlich nicht bietet, stellt sich die Frage nach dessen Beitrag. Wenn man eine grobe Einordnung treffen möchte, so werden Sicherheitskonzepte sowie bekannte Schwachstellen der folgenden beiden Säulen bedient:

I. Drahtlose Übertragungstechnologien
II. Softwarekomponenten mobiler digitaler Geräte

Es wird sich aber sehr schnell zeigen, dass solch eine strikte Klassifizierung oftmals nicht durchhaltbar ist. Beim Verfassen dieses Buches stellte sich zudem heraus, dass es häufig einfacher ist, die Verwundbarkeit einzelner Komponenten zu beschreiben, als über deren gelungene und im Idealfall beweisbar sichere Sicherheitsarchitektur zu berichten. Im Vordergrund steht daher vor allem die Vermittlung eines umfassenden Problembewusstseins

und die Erörterung der Fragestellung, warum es eben nicht so einfach oder nahezu unmöglich ist, komplexe IT-Systeme abzusichern. Hierzu muss man sich eigentlich nur anschauen, aus wie vielen Seiten Spezifikation einzelne Standarddokumente der Internet Engineering Task Force (IETF) bzw. des 3rd Generation Partnership Project (3GPP) oder des Europäischen Instituts für Telekommunikationsnormen (ETSI) bestehen. Oder aus wie vielen Lines-of-Code Bluetooth, WLAN und andere drahtlose Kommunikationsprotokolle bestehen bzw. wie ‚geschwätzig' heutige Betriebssysteme sind, was den ungefragten Aufbau von Verbindungen zu IP-Adressen in alle Welt angeht. Denn dann erhält man einen Eindruck von der allumfassenden Problematik, mit der sich nicht nur der IT-Sicherheitsbeauftragte und die IT-Abteilung eines Unternehmens auseinanderzusetzen haben. Nicht zu vergessen sind die vielfältigsten Aspekte und gegenseitigen Abhängigkeiten, die sich zum Teil aus der, in Standards vorgeschriebenen Abwärtskompatibilität einzelner kryptografischer Verfahren für Protokollklassen zur Drahtlos-Kommunikation ergeben oder ganz einfach durch offen formulierte respektive via *hidden agenda* ausgetragene Zielkonflikte einzelner Interessengruppen innerhalb derartiger Standardisierungsgremien.

Die Sorglosigkeit, mit der wir alles miteinander vernetzen, sei es in den Bereichen Industrie 4.0, Smart Home, Advanced Metering Infrastructure (AMI), oder im Bereich des autonomen Fahrens und der Fahrzeug-zu-Fahrzeug-Kommunikation macht zumindest mich schier fassungslos. Wie können wir annehmen, dass aus einer Reihe von nachweisbar unsicheren Komponenten ein sicheres und gegenüber verschiedensten Arten von Angreiferkategorien gehärtetes Gesamtsystem entsteht? Zauberformeln wie *‚Predictive Maintenance'* oder *‚Mensch, Maschine, Werkstück.., alles digital vernetzt'* sollen gerade auch den deutschen Mittelstand bewegen, seine Produktionsstätten zu vernetzen, oftmals animiert durch weltweit agierende internationale Konzerne, deren Zulieferer sie sind.

So kann es bei aller Euphorie um die digital Vernetzung nicht schaden, einen Schritt zurückzugehen und sich die Diskussionen in ‚artfremden' Disziplinen vor Augen zu führen, in denen das Thema Sicherheit eine deutlich längere Tradition hat: Im Beitrag *‚Sicherheitsforschung – Sichern auf Hochtour'* [4] wird für das alpine Gehen neben verschiedenen Seilsicherungsformen mit Seil auch das *seilfreie Gehen* als Option erörtert. Hierzu heißt es: *„Dem bewussten Verzicht auf ein Sicherungsseil liegt eine nüchterne Risikoabwägung zugrunde: Das Schadensausmaß ist reduziert, wenn nur eine Person ins Rutschen kommt. … Wenn dagegen eine angeseilte Seilschaft mal Fahrt aufgenommen hat, verheddern sich die Mitglieder im Seil und ziehen sich gegenseitig nach unten."* Und weiter: *„Die Abwägung, wann seilfreies Gehen noch für alle im Team passt, ist nicht einfach. Sie erfordert realistische Selbsteinschätzung, Einfühlungsvermögen und offene, klare Kommunikation. Wie generell jede Entscheidung über angemessene Sicherungsmaßnahmen."*

Es ist nicht zuletzt diese Forderung einer klaren und ungeschönten Kommunikation über angemessene Sicherungsmaßnahmen, die die IT-Welt beherzigen und verinnerlichen sollte. Dies betrifft insbesondere die Unternehmensführung, die meist nicht mit *‚Bits and Pieces'* behelligt werden möchte. Wer nun der Meinung ist, dass der obige Vergleich doch arg hinkt, der möge sich insbesondere mit den Themen dieses Buches auseinandersetzen. Übrigens: In der IT-Welt wären mit *‚seilfreiem Gehen'* die *‚air-gapped systems'* gemeint, also solche, die keinerlei Kommunikationsschnittstelle miteinander teilen.

1.3 Danksagung

Das vorliegende Buch wäre in seiner jetzigen Form nicht denkbar gewesen ohne die tatkräftige praktische Unterstützung einer ganzen Reihe von engagierten UNITS- und ENITS-Studierenden der Hochschule Offenburg. So haben insbesondere Max Bauert, Alexander Eger, Jan Breig und Moritz Kaumanns, aber auch andere Studierende verschiedenste Praxistests in der Laborumgebung durchgeführt. Darüber hinaus hat Frau Dr. Gisela Hillenbrand wertvolle Anregungen zur Verbesserung der Lesbarkeit zum Abschnitt Angriffe über ‚ungültige Kurven' beigetragen. Ohne die Arbeiten des Doktoranden Louis Tajan zur Anwendung von Bloom-Filtern wäre der Abschnitt zum Thema Auskunftsansprüche mit technischem Datenschutz in seiner jetzigen Form nicht entstanden. Ihnen allen gilt mein besonderer Dank!

1.4 Noch eine Bemerkung in eigener Sache

Bevor es los geht, noch eine Bemerkung in eigener Sache:

Wir wollen Schwachstellen verstehen und Angriffsszenarien nachvollziehen!

Wichtig dabei ist:

- Der Inhalt des Buches soll nicht als „Hacker-Anleitung" dienen.
- Der Inhalt des Buches soll insbesondere nicht dazu beitragen, sich auf illegalem Wege Daten zu beschaffen

Vorsicht
Die Anwendung der gezeigten Techniken und Methoden kann gegen geltendes Recht verstoßen!
Im Einzelnen sind dies:

- StGB § 202a: Ausspähen von Daten
- StGB § 202b: Abfangen von Daten
- StGB § 202c: **„Der Hackerparagraph"**
- StGB § 303a: Datenveränderung
- StGB § 303b: Computersabotage

Prof. Dr. habil. Dirk Westhoff vertritt an der Hochschule Offenburg das Themengebiet Sicherheit und Verlässlichkeit in Informationssystemen. Er ist Gründungsmitglied des Institutes für verlässliche Embedded und Kommunikationselektronik (ivESK) und Studiendekan des Master-Studienganges ENITS (Enterprise- and IT-Security). Vorherige Stationen an Hochschulen für Angewandte Wissenschaften waren Hamburg und

Furtwangen. Dr. Westhoff habilitierte 2007 zum Thema Sicherheit und Verlässlichkeit drahtloser Zugangsnetze an der FernUniversität Hagen. Im Unternehmen NEC R&D am Standort Heidelberg war er zuständig für die Einwerbung und Durchführung von EU-Projekten wie EU FP6-IST STREP UbiSec&Sens[1] (Technischer Projektleiter), EU FP7-IST SENSEI[2] sowie EU FP7-IST STREP WSAN4CIP[3], bei denen ein Schwerpunkt auf der Erarbeitung von Sicherheitslösungen für drahtlose Sensornetze lag. Projekte aus Hamburger Zeiten sind SKIMS[4] (BMBF) und Smart Power Hamburg[5] (BMWI). In Furtwangen und Offenburg erfolgten Arbeiten auf den BMBF-Projekten UNIKOPS[6] und ProSeCCo[7] sowie dem Projekt PAL-SAaaS[8], wobei UNIKOPS sich der Erarbeitung universell konfigurierbarer Sicherheitslösungen für Cyber-physische Systeme widmete. Dr. Westhoff ist Mitbegründer der Springer-Serie LNCS ESAS[9] und Co-Autor von ca. 90 Veröffentlichungen. Dr. Westhoff hält acht Patente zu Sicherheit in drahtlosen Sensornetzen sowie in verteilten Systemen.

Literatur

1. Brunner, M., Hofinger, H., Krauß, C., Roblee, C., Schoo, P., Todt, S.: Infiltrating critical infrastructures with next-generation attacks: W32.Stuxnet as a showcase threat. Fraunhofer SIT, Darmstadt. https://www.sec.in.tum.de/i20/publications/infiltrating-critical-infrastructures-with-next-generation-attacks-w32-stuxnet-as-a-showcase-threat (2010). Zugegriffen: 26. Apr. 2019
2. Chekoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czekis, A., Roesner, F., Kohno, T.: Comprehensive experimental analyses of automotive attack surfaces. In: Proceedings of the 20th USENIX Conference on Security, SEC'11, San Francisco (2011)
3. Data Breach Investigations Report: www.verizonenterprise.com/de/DBIR/2015/ (2015). Zugegriffen: 1. Dez. 2015
4. Hellberg, F.: Sicherheitsforschung – Gletscher: Seil oder nicht Seil? DAV Panor. 3, 68–71 (2018)

[1]UbiSec&Sens – Ubiquitous Sensing and Security in the European Homeland.

[2]SENSEI – Integrating the Physical with the Digital World of the Network of the Future.

[3]WSAN4CIP – Wireless Sensor and Actuator Networks for Critical Infrastructure Protection.

[4]SKIMS – Schichtenübergreifendes kooperatives Immunsystem für mobile, mehrseitige Sicherheit.

[5]SmartPower Hamburg – Security for smart grid IT architectures.

[6]UNIKOPS – Universell konfigurierbare Sicherheitslösung für Cyber-physische heterogene Systeme.

[7]ProSeCCo – Promotionsvorhaben zur Erarbeitung von Sicherheitserweiterungen für das Cloud Computing.

[8]PAL-SAaaS – Building Triangular Trust for Secure Cloud Auditing.

[9]ESAS – European Workshop on Security in Ad Hoc & Sensor Networks.

Teil I

Grundlagen

2 Wissenswertes zu Netzen, sowie mathematische und kryptografische Hintergründe

Für ein besseres Verständnis bei der Analyse von Schwachstellen mobiler Systeme, aber auch um einzelne Angriffsszenarien besser nachvollziehen zu können, ist es ratsam sich mit ausgewählten Themen der Kryptografie, einigen zahlentheoretischen Aspekten, aber auch fundamentalen Aspekten zu Netzwerken und Protokollverhalten vertraut zu machen bzw. sich diese noch einmal zu vergegenwärtigen. Dabei sollen die Themen so knapp wie möglich jedoch so umfassend wie nötig aufbereitet werden.

2.1 Anmerkungen zur Notation

Es haben sich zur Beschreibung von Sicherheitsprotokollen einige Notationen etabliert und bewährt, die wir der Vollständigkeit halber an dieser Stelle nennen und im Verlaufe dieses Buches einsetzen werden.

Spricht man von A, B und E, bzw. Alice, Bob und Eve, so ist die Kommunikation zwischen zwei Parteien Alice und Bob gemeint, die von einem *passiven* Angreifer Eve belauscht (engl. *eavesdropping*) wird. Eve kann jegliche übertragenen Nachrichten a, b in beide Übertragungsrichtungen abhören, ist aber nicht in der Lage, die Nachrichten abzuändern oder zu blockieren (Abb. 2.1).

Spricht man hingegen von A, B und M, bzw. Alice, Bob und Mallory, so ist die Kommunikation zwischen zwei Parteien Alice und Bob gemeint, die von einem *aktiven* Angreifer Mallory gestört (engl. *malicious*) wird. Mallory kann jegliche Nachrichten a, b in beide Richtungen mithören und verwirft oder ändert diese je nach Bedarf in a' und/oder b' (Abb. 2.2). Je nach Pfeilrichtung bedeutet $x|y$ also, dass die Nachricht x von Mallory in y geändert wird (bei $\rightarrow$) oder die Nachricht y in x geändert (bei $\leftarrow$) wird.

Darüber hinaus hat sich noch eine weitere Notation etabliert, wenn man aufzeigen will, welche Partei im Verlaufe des Angriffs korrumpiert werden soll. Diese wird oftmals

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_2

A(lice)		E(ve)		B(ob)
a	$\rightarrow$	a	$\rightarrow$	a
b	$\leftarrow$	b	$\leftarrow$	b

Abb. 2.1 Notation einer Kommunikation mit passivem Angreifer (E)ve. (Quelle: eigene Darstellung)

A(lice)		M(allory)		B(ob)
a	$\rightarrow$	$a\|a'$	$\rightarrow$	a'
b'	$\leftarrow$	$b'\|b$	$\leftarrow$	b

Abb. 2.2 Notation einer Kommunikation mit aktivem Angreifer (M)allory. (Quelle: eigene Darstellung)

O(pfer)		M(allory)		B(ob)
a	$\rightarrow$	$a\|a'$	$\rightarrow$	a
b'	$\leftarrow$	$b'\|b$	$\leftarrow$	b

Abb. 2.3 Notation einer Kommunikation mit aktivem Angreifer sowie dem zu korrumpierenden O(pfergerät). (Quelle: eigene Darstellung)

als Opfer(gerät) bezeichnet, wodurch dokumentiert wird, welchem Ziel(gerät) die Manipulation von Nachrichten durch Mallory dient (Abb. 2.3).

Während mit den obigen eingeführten Notationen die Fähigkeiten eines passiven Angreifers E(ve), der sich in Übertragungsreichweite von Alice und Bob befindet, klar umrissen sind, stellt sich in den nun folgenden Abschnitten heraus, dass dies für die angenommenen Fähigkeiten des aktiven Angreifers M(allory) nicht immer so eindeutig ist. Die zugrunde gelegten Fähigkeiten von Mallory sind von Angriff zu Angriff teilweise sehr unterschiedlich, wie wir sehen werden. Das Opfer(gerät) verhält sich im Sinne einer *Blackbox* immer so, wie es die Spezifikation vorsieht, oder -besser- wie die Spezifikation durch die vorgefundene konkrete Implementierung umgesetzt wurde.

Manchmal ist es hilfreich, gerade auch im Falle von drahtloser Kommunikation die Unterscheidung zu treffen, ob Daten mitgehört werden E(ve) oder aber nicht nur mitgehört, sondern immer über eine Partei, den Angreifer-(P)roxy geleitet werden müssen (Abb. 2.4).

Dies mag notwendig sein damit die Daten überhaupt zu Alice und Bob gelangen. Bei einem Angreifer Eve ist dies nicht der Fall, der zwar auf dem drahtlosen Übertragungsmedium mithört, jedoch die Nachrichten nicht weiterleitet.

A(lice)		P(roxy)		B(ob)
a	$\rightarrow$	a	$\rightarrow$	a
b	$\leftarrow$	b	$\leftarrow$	b

Abb. 2.4 Notation einer Kommunikation mit passivem (P)roxy. (Quelle: eigene Darstellung)

Innerhalb dieses Buches bedeuten die Pfeile „→“ bzw. „←“ das erfolgreiche Senden und gleichzeitig auch erfolgreiche Empfangen einer Nachricht. Soll dagegen aufgezeigt werden, dass eine Nachricht zwar gesendet, jedoch nicht erfolgreich empfangen wurde, so verwenden wir in Abhängigkeit von den Positionen der Quelle und der Senke die Darstellungen „→!“ oder „!←“. Eine Nachricht hat immer den Aufbau Typ_{S}[a](p). Dabei beschreibt „Typ“ den Nachrichtentyp, „S“ den verwendeten Schlüssel sofern er vorhanden ist, „a“ den Adressaten und „p“ ist schließlich der Inhalt der zu versendenden Nachricht. Wird eine Nachricht an alle Teilnehmer im lokalen Netz bzw. der Übertragungsreichweite versendet (Engl. *broadcasting*), so wird dies mit „bc“ notiert. Nur wenn ein Schlüssel „S“ tatsächlich verwendet wird, wird dieser auch in die Darstellung aufgenommen. Oftmals ist darüber hinaus noch festzulegen, welcher *Kommunikationsschicht* ein Nachrichtentyp zuzuordnen ist. Dies ist dann dem Fließtext zu entnehmen. Was genau eine Kommunikationsschicht ist wollen wir im nachfolgenden Abschnitt verdeutlichen.

2.2 Wissenswertes zu Netzen und Protokollen

Einige allgemeine Aspekte aus dem Bereich der Netzwerke und des Protokollentwurfs zur Übertragung von Nachrichten sollten wir uns vorab vergegenwärtigen, bevor wir dann später dedizierter in eine Schwachstellenanalyse mobiler Systeme einsteigen. Wir beginnen daher zunächst damit, uns zu verdeutlichen, was man unter einem Protokoll in Kommunikationsnetzen versteht, und erörtern anschließend das bekannte Zwei-Armeen-Problem, welches ein fundamentales Dilemma von Kommunikationsprotokollen offenlegt. Im praktischen Teil (Abschn. 2.2.2) lernen wir einige hilfreiche unixoide Shell-Kommandos kennen mit denen Sie die Erreichbarkeit eines Gerätes in IP-Netzen prüfen können, bzw. ein elementares Verständnis über das Zusammenwirken einzelner Netzwerkprotokolle erlangen können.

2.2.1 Protokolle

Ähnlich wie das (Empfangs-)Protokoll bei Besuchen von Staatsoberhäuptern, Regierungschefs oder ganz allgemein festlichen Empfängen den Ablauf eines Empfangs sehr detailliert regelt, so werden für deren Einsatz in Netzwerken Protokolle definiert, die ebenfalls die Kommunikation von Parteien regeln. Nur dass hierbei die

kommunizierenden Parteien eben keine Personen darstellen, sondern Rechner, Programme oder besser *Prozesse* welche auf verschiedenen Systemen zur Ausführung gelangen und Nachrichten entgegennehmen, diese verarbeiten und bei Bedarf selber wieder Nachrichten versenden. Doch bevor wir nun auf derartige Regelwerke zur Kommunikation zwischen Prozessen auf verschiedenen, durch ein Netzwerk verbundenen Rechnern eingehen, vorab noch ein Beispiel dafür, dass für eine erfolgreiche und zielführende Kommunikation zwischen Personen auch die Berücksichtigung des kulturellen Hintergrundes vorteilhaft sein kann. So mag der eine oder andere Leser schon einmal verwundert das Gesprächsverhalten von, beispielsweise, Mitteleuropäern, Südeuropäern und Japanern verfolgt haben. Für einen Mitteleuropäer erscheint das Gespräch zwischen Südeuropäern als ein sich gegenseitig ständiges Ins-Wort-Fallen. Noch bevor die eine Seite ausgeredet hat, fängt die andere Seite schon wieder an zu reden. Doch irgendwie funktioniert der Austausch dennoch. Kommunizieren Japaner miteinander, so ist häufig ein völlig anderes Kommunikationsmuster zu beobachten: Der Gesprächspartner fällt dem Gegenüber unter keinen Umständen ins Wort. Mehr noch: Zum Zeichen der Abwägung seiner Worte und des Gesagten lässt er eine gewisse Zeit verstreichen, bis er selbst das Wort ergreift. So können bis zu einer Antwort durchaus einige Sekunden der Stille vergehen. Aus der Perspektive eines Mitteleuropäers können nun insbesondere in Telefonaten, bei denen die Gesprächspartner nicht mit nonverbalen Gesten andeuten könnten, welche der beiden Seiten nun ‚an der Reihe' ist, mit Personen solch unterschiedlicher soziokultureller Kreise durchaus eine spannende Herausforderung sein: In dem einen Fall tut sich der Mitteleuropäer vermutlich schwer sich selber Gehör zu verschaffen. Er wird ständig unterbrochen oder aber kommt erst gar nicht zu Wort. In dem anderen Fall kann er seinen Standpunkt zwar vortragen, es wird ihm sicher aber schwerfallen, die einsetzende, für sein Empfinden oftmals zu lange Stille nach der Beendigung seiner Sätze zu ertragen. Der Mitteleuropäer wird das Schweigen der Gegenseite nun häufig als Nichtverstehen oder gar Missbilligung des Gesagten auslegen. Er wird aus diesem Grund vermutlich das Schweigen seines Gegenübers unterbrechen, um noch eine Erklärung nachzuschieben oder gar sein Gesagtes zu relativieren. Mit diesem Verhalten ist er nun aber für das Empfinden des asiatischen Gesprächspartners unhöflich, da dieser nun gar nicht zu Wort kommt.

Warum schildere ich diese Beobachtungen in einem Buch, in dem es doch um die Vermittlung von Technikverständnis gehen soll? Nun, der beiderseitige und im Idealfall ähnliche Kommunikationshintergrund der Kommunikationsparteien scheint bedeutsam für einen erfolgreichen Austausch von Nachrichten und Informationen zu sein. Diese Beobachtung um die Vorteilhaftigkeit eines ähnlichen Kommunikationshintergrundes wurde auch beim Entwurf von Protokollen für technische Kommunikationsnetze berücksichtigt. So hat es sich bewährt bei einem Protokollentwurf in *Kommunikationsschichten* zu denken, wobei die jeweils untere Schicht der auf ihr aufsetzenden Schicht bestimmte Dienste zu Verfügung stellt. Daher sprechen wir innerhalb der Disziplin Kommunikations- und Rechnernetze beispielsweise von einem Protokoll mit Namen X_1 welches auf der Schicht y angesiedelt ist und beispielsweise auf einem Protokoll X_2 der Schicht

$y-1$ aufsetzt. Die übertragenen Daten würden sich dann aus den einzelnen wissenswerten Informationen zum Protokoll X_1, die in einem $Kopf_{X1}$ enthalten sind, Informationen zum Protokoll X_2, die in einem $Kopf_{X2}$ enthalten sind, sowie den eigentlichen zu übertragenden Nutzdaten zusammensetzen, also $Kopf_{X1}, Kopf_{X2}, Nutzdaten$. Dabei ist es durchaus üblich, dass das Protokoll X_2 seinerseits wiederum auf Protokollen aufsetzt. Auch das Protokoll X_1 könnte seine Dienste wiederum einem Protokoll das auf der Schicht $y+1$ angesiedelt ist, bereitstellen. Dieses würde sich dann auf die Anzahl der in einer Dateneinheit enthaltenen unterschiedlichen Köpfe auswirken.

Ein *Protokoll* legt nun also genaue Regeln der Abfolge fest nach denen beide Seiten sich zu verhalten haben. Ein regelkonformes Verhalten begünstigt, bzw. macht den beidseitigen geordneten Informationsaustausch erst möglich. So enthalten Kommunikationsprotokolle oftmals eine Phase der Begrüßung, auch bekannt als *Handshake*-Phase, die der beiderseitigen Bekanntmachung der Parteien, dem Aufbau einer Verbindung, sowie der Abstimmung weiterer, für die nachfolgende Kommunikation, erforderlichen Parameter dient. Ebenfalls muss vorab geklärt sein welche der Parteien sich im zuhörenden Modus befindet und von welcher Seite erwartet wird etwas zu senden. Ist dies gegeben, so senden Alice und Bob sogenannte *Hello*-Nachrichten. Alice und Bob sind nunmehr Prozesse, also sich in der Ausführung befindende Programme. Diese können wahlweise auf einem PC, Laptop, einem Smartphone oder anderen Geräten mit einer integrierten Recheneinheit gestartet sein.

So beinhaltet eine Nachricht vom Typ *client hello,* die von einem Prozess versendet wird beispielsweise Informationen, wer die anfragende Partei ist, und über welche Möglichkeiten der Client für die weitere Kommunikation verfügt. Bei einer normalen Abfolge der Kommunikation sendet der Server eine Nachricht vom Typ *server hello* zurück. Mit dieser Nachricht signalisiert der Server seine Bereitschaft nun auch weitere Nachrichten entgegennehmen zu wollen (Abb. 2.5). Je nach konkreter Ausgestaltung des Handshake-Protokolls teilt der Server dem Client in der Nachricht vom Typ *server hello* darüber hinaus noch mit welche der vom Client angebotenen Möglichkeiten der weiteren Kommunikation der Server ausgewählt hat. Solch ein Verhalten ist typisch für die erste Phase von *verbindungsorientierten* Protokollen. Darüber hinaus existieren zu einem Protokoll noch Nachrichtentypen mit denen eine Seite einen geordneten Verbindungsabbau einleiten kann, oder oftmals auch Möglichkeiten mit der eine der beiden

Abb. 2.5 Einfaches, fiktives Handshake-Protokoll zwischen Client und Server. (Quelle: eigene Darstellung)

Kommunikationsparteien bei einer bestehenden Verbindung weitere Unterprotokolle starten oder ändern kann. Und wäre dies nicht schon kompliziert genug so kennt das heutige Internet eine große Menge von Protokollen, allen voran IPv4 und IPv6, das *Transmission Control Protocol* (TCP), das *User Datagram Protocol* (UDP), das *Internet Control Message Protocol* (ICMP), http(s), arp und natürlich auch dedizierte Protokolle zur drahtlosen Übertragung von Nachrichten. Auf einige davon werden wir im Verlaufe dieses Buches noch genauer zu sprechen kommen.

2.2.2 Aus der Praxis

Unixoide Systeme bieten eine ganze Reihe von Befehlen mit denen man einen ersten Eindruck über den aktuellen Netzwerkzustand sowie das Protokollverhalten erlangen kann. So können Sie mit dem Kommando `ping` testen ob ein Rechner mit einer bestimmten IP-Adresse aktuell über das Netz erreichbar ist. Der Befehl `ping` nutzt hierfür das Internet Control Message Protocol (ICMP), um Datagramme zu versenden. Neben der Tatsache, ob ein Rechner erreichbar ist, wird auch die Paketumlaufzeit *(round-trip-time)* für jedes gesendete Paket angezeigt. Dies ist die für den Hin- und Rücktransport über das Netzwerk benötigte Zeit. Der Befehl `traceroute` setzt ebenfalls auf das Protokoll ICMP auf. Führen Sie dieses Kommando über eine Shell aus, so ist es Ihnen möglich einige weiterführende Informationen zum Weg und den hierzu besuchten Zwischenstationen der Pakete durch ein IP-Netz zu erhalten bis diese schlussendlich den adressierten Zielrechner erreichen.

Der Befehl `netcat` oder einfach `nc` dagegen verwendet nicht ICMP, sondern setzt auf Protokollen der Transportschicht auf. In der Voreinstellung wird das oben bereits erwähnte Protokoll TCP verwendet. Optional ist `netcat` auch über UDP verwendbar. Gibt man bei `netcat` den Rechner sowie den adressierten Port (im Beispiel 11111) an, so wird mittels `netcat` eine TCP-Verbindung etabliert, über die Sie über die Standardeingabe der Shell Daten eingeben und versenden können. Haben Sie im Vorfeld mit `netcat` unter Verwendung der Option `-l` im zuhörenden Modus einen Serverprozess mit der gleichen Portnummer etabliert, so können Sie zwischen diesen beiden Systemen nun Daten versenden. Man kann sich mit der Funktionsweise von `netcat` vertraut machen, indem man auf einem System zwei Shells öffnet und als Host jeweils `localhost` (127.0.0.1) mit der gleichen Portnummer verwendet. In der ersten Shell rufen Sie demnach auf:

```
$ netcat -l -p 11111 127.0.0.1
```

Der Aufruf aus der zweiten Shell auf dem gleichen System könnte dann wie folgt aussehen:

```
$ netcat 127.0.0.1 11111
```

Nun können Sie beispielsweise Eingaben wie „`Diesen Text übertrage ich nun`" mittels des Protokolls TCP versenden und in einer Shell zur Anzeige bringen. Versuchen Sie es. Sind zwei Rechner in dem gleichen IP-Netz zugegen und erreichbar, so kann man wie oben vorgehen, indem man `localhost` durch die entsprechenden IP-Adressen der Zielsysteme ersetzt.

Hat man darüber hinaus auch das Netzwerk-Analysewerkzeug *Wireshark* zur Hand, so lassen sich die tatsächlich versendeten Daten herausfiltern. Hierzu sollten sie den Filter in Wireshark im Zuge des obigen Beispiels auf `tcp.port==11111` gesetzt haben. Mit ein wenig Übung und Geduld sollte es Ihnen möglich sein, mit Hilfe von Wireshark die von Ihnen mittels `netcat` über das Transportprotokoll TCP versendeten Daten herauszufiltern. Haben Sie Ihren Text `Diesen Text übertrage ich nun` gefunden? Gleichzeitig werden Sie sehen wie viele andere Nachrichten im Rahmen der TCP Verbindung zwischen den Systemen ausgetauscht werden. Einige davon betreffen den TCP-*Handshake.* Darüber hinaus wird ersichtlich, dass TCP nicht das einzige ausgeführte Protokoll ist, denn dieses nutzt wiederum weitere Protokolle auf unteren Schichten, damit Ihr Text auch tatsächlich wohlbehalten und weitgehend unabhängig von dem zur Verfügung stehenden Übertragungsmedium ans Ziel gelangt. Man mache sich klar: Die zu überbrückenden Teilstrecken (siehe Funktionsweise `traceroute`) können teilweise drahtgebunden, aber natürlich auch drahtlos sein, womit es zu teilweise erheblichen Unterschieden hinsichtlich der Übertragungsqualität und Datenverlusten auf den einzelnen Teilstrecken kommen kann. Dies alles so auszugleichen und zu berücksichtigen, dass TCP ihren erzeugten Text mit einer sehr hohen Wahrscheinlichkeit erfolgreich auch über ein rauschbehaftetes Medium wie Luft übertragen kann, obliegt zu einem großen Teil den eingesetzten Protokollen der unteren Schichten, aber natürlich auch dem Protokoll TCP selbst.

▶ Machen Sie sich bitte klar, dass wenn Sie mit `Ctr-z` den Eingabeprompt in der Shell, in der Sie `netcat` gestartet haben zurückholen, dies nicht gleichbedeutend ist mit der Beendigung des Prozesses, der `netcat.exe` ausführt. Oder, ein wenig konkreter: Solange Sie den `netcat`–Prozess nicht sauber terminiert haben, läuft dieser Prozess noch eine Weile im Hintergrund und Ihr Rechner ist auch weiterhin über die TCP-Verbindung von anderen Rechnern, potenziell weltweit, ansprechbar! Dies ist ein Sicherheitsrisiko insbesondere dann, wenn Sie netcat mit weiteren Optionen wie -e zum Ansprechen von weiterem ausführbaren Code gestartet haben! Sie sollten daher unbedingt beispielsweise unter Aufruf von `ps -a` eruieren, unter welcher Prozess-ID `(UID) netcat` auf Ihrem System gegenwärtig noch läuft. Haben Sie dies herausgefunden, so löschen Sie diesen Prozess mittels `Ctr-C` aus der Shell heraus oder aber mittels `kill -9` und der Angabe derjenigen `UID`, die zum `netcat` Prozess gehört. Mittels `pstree` oder aber nochmals mit `ps -a` können Sie dann überprüfen ob der Prozess nun tatsächlich terminiert ist.

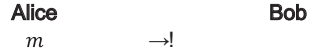

Abb. 2.6 Übertragungsfehler beim Versenden der eigentlichen Nachricht *m*. (Quelle: eigene Darstellung)

2.2.3 Zwei-Armeen-Problem

Sendet nun Alice eine Nachricht an Bob und erwartet von diesem aufgrund einer Regel im beidseitig zum Einsatz kommenden Protokoll eine Bestätigung für den Erhalt dieser Nachricht, so ist aufgrund möglicherweise auftretender Übertragungsfehler auf der Übertragungsstrecke nicht immer sichergestellt, dass die Nachricht auch tatsächlich beim Empfänger ankommt. Da dies sowohl für das Senden der eigentlichen Nachricht als auch für das Senden einer Bestätigungsnachricht gleichermaßen eintreten kann, hat Alice bei einer nicht eintreffenden Bestätigungsnachricht von Bob keinerlei Kenntnis darüber welche der beiden Nachrichten nun tatsächlich nicht angekommen ist. Diese beiden, für Alice nicht unterscheidbaren Fälle, sind in den Abb. 2.6 und 2.7 dargestellt.

Dieses Dilemma hat in der Fachliteratur unter dem Begriff ‚Zwei-Armeen-Problem' eine gewisse Berühmtheit erlangt. Zwei schwächere Armeen sind bestrebt, einen stärkeren Gegner nur gleichzeitig anzugreifen, damit das Unterfangen für diese Seite auch tatsächlich siegreich ausgehen kann. Denn gemeinsam sind sie dem stärkeren Gegner überlegen, allein jedoch in der Unterzahl. Um sich derart abzustimmen, müsste jedoch ein Bote mit einer Nachricht, in der der Zeitpunkt für den gleichzeitigen Angriff festgelegt ist, die feindlichen Linien überwinden. Allerdings kann man nicht davon ausgehen, dass der Bote auch tatsächlich unbeschadet die feindlichen Linien passieren kann und die Nachricht übermitteln konnte. Mit dieser Ungewissheit um das Schicksal des Boten wird es für die schwächere Armee, welche den Boten gesendet hat, immer ein enormes Risiko sein die stärkere Armee anzugreifen, wohlwissend, dass die kollaborierende Armee aufgrund dessen, dass sie die Nachricht nicht erhalten haben könnte eben nicht gleichzeitig in die Schlacht zieht. Diese Ausführung und die sich nun anschließende Beweisskizze entstammen [7].

Doch zurück zu Alice und Bob: Soll Alice die Nachricht *m* noch einmal senden, unter der Annahme, dass *m* Bob nicht erreicht hat? Oder hat Bob die Nachricht bereits erhalten und es ist lediglich die Bestätigung $ACK(m)$ des Erhaltens der Nachricht *m* nicht übertragen worden? Tatsächlich kann man beweisen, dass es kein wie auch immer geartetes Protokoll gibt, um dieses Dilemma zu beheben. Die Beweisidee ist recht simpel: Wäre

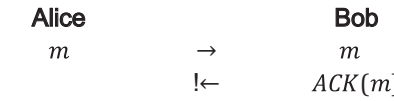

Abb. 2.7 Übertragungsfehler beim Versenden der Bestätigung $ACK(m)$ zum Erhalt der Nachricht *m*. (Quelle: eigene Darstellung)

die letzte zu sendende Nachricht einer wie auch immer gearteten Abfolge von Nachrichten unwesentlich, so kann sie auch weggelassen werden. Ist sie es nicht, so ist es für die erfolgreiche Abarbeitung des Protokolls unabdingbar, dass diese Nachricht ankommt. In gleicher Weise lässt sich für die vorletzte Nachricht, die vorvorletzte und so weiter argumentieren.

Praktische Umsetzungen von Protokollentwürfen verwenden daher einen Zeitbegrenzer (*timer*) um das geschilderte Problem ein wenig abzuschwächen. Vergeht eine gewisse Zeitspanne ohne, dass bei Alice die Bestätigung eingegangen ist, so sendet Alice die Nachricht m ein weiteres Mal ohne letztendlich entscheiden zu können, welche Nachricht tatsächlich verloren gegangen ist.

Warum ist das Zwei-Armeen-Problem nun für uns interessant, wo es doch eigentlich unser Bestreben ist, uns mit der Sicherheit bzw. der Verwundbarkeit mobiler Systeme auseinanderzusetzen? Wir werden im Verlaufe dieses Buches feststellen, dass diese dem Zwei-Armeen-Problem inhärente Unentscheidbarkeit von findigen Sicherheitsexperten bei einigen drahtlosen Kommunikationssystemen ausgenutzt wurde, um Sicherheitsprotokolle auszuhebeln. Dazu mussten weder die im Protokoll verwendeten kryptografischen Bausteine gebrochen werden noch musste der verwendete Schlüssel dem Angreifer bekannt sein.

2.3 Ausgewählte zahlentheoretische Aspekte

Einige der in diesem Buch vorgestellten Verwundbarkeiten von Sicherheitsprotokollen in mobilen Systemen sind in ihrer Gänze nur zu verstehen, wenn man sich ein wenig eingehender mit ausgewählten algebraischen sowie zahlentheoretischen Aspekten vertraut macht. Dies soll an dieser Stelle geschehen, wohlwissend, dass so mancher Leser, der sich als zukünftiger Sicherheitsexperte oder ‚Pentester' versteht, sich eher zähneknirschend mit diesen Dingen beschäftigen möchte, da die *low hanging fruits* ja auch oftmals ohne derartiges mathematisches Verständnis zu ernten sind. Dennoch folgen hier nun einige mathematische Grundlagen die an späterer Stelle des Buches dazu dienen, einzelne der vorgestellten Sicherheitslösungen für drahtlose Übertragungsprotokolle besser einordnen zu können oder diese gar zu brechen.

2.3.1 Kongruenz

Gegeben seien zwei ganze Zahlen $a, b \in \mathbb{Z}$. Man spricht davon, dass die Zahl a kongruent zu b modulo n ist, genau dann, wenn die Zahl n die Zahl $(a - b)$ teilt. Die Notation hierfür ist

$$a \equiv b \ (mod\ n)$$

Die Zahl n ist der Modulus der Kongruenz.

2.3.2 Ganzzahliger Ring

Ein ganzzahliger Ring ist eine algebraische Struktur, die auf der oben beschriebenen modularen Arithmetik beruht. Die Menge $\mathbb{Z}_m = \{0, 1, 2, \ldots, \mathrm{m} - 1\}$ bildet zusammen mit der additiven Operation $+$ und der multiplikativen Operation $\cdot$ einen Ring, wobei für alle Elemente $a, b \in \mathbb{Z}_m$ gilt:

$$a + b \equiv c \ (mod\ m), \quad (c \in \mathbb{Z}_m)$$

$$a \cdot b \equiv d \ (mod\ m), \quad (d \in \mathbb{Z}_m)$$

Man mache sich klar: Die modulare Arithmetik bietet einen einfachen und eleganten Weg, Arithmetik auf einer endlichen Menge von Zahlen durchzuführen. So bleiben bei einem ganzzahligen Ring $\mathbb{Z}_m$ mit den Operationen Addition $+$ und Multiplikation $\cdot$ für alle $a, b \in \mathbb{Z}_m$ auch deren Ergebnisse wieder innerhalb der Menge $\mathbb{Z}_m$. Diese Eigenschaft wird als *Abgeschlossenheit* bezeichnet. Daneben existiert genau ein *neutrales Element* 0 *für die Addition.* Für alle $a \in \mathbb{Z}_m$ ergibt die Addition mit dem additiven neutralen Element $a + 0 \equiv a \ (mod\ m)$. Jedes Element $a \in \mathbb{Z}_m$ verfügt über ein *additives Inverses* $-a$. Es gilt $a + (-a) \equiv 0$. Für die Operation der Multiplikation lassen sich darüber hinaus die folgenden Ringeigenschaften nennen: Es existiert ein *neutrales Element* 1 *für die Multiplikation,* sodass für alle $a \in \mathbb{Z}_m$ gilt $a \cdot 1 \equiv a \ (mod\ m)$. Allerdings existiert nur für einige Elemente $a \in \mathbb{Z}_m$ das *multiplikative Inverse* a^{-1}, sodass $a \cdot a^{-1} \equiv 1 \ (mod\ m)$ gilt. Damit lassen sich zwei Fragen zu einem Element $a \in \mathbb{Z}_m$ formulieren:

1. Existiert das multiplikative Inverse a^{-1} zu dem Element a?
2. Falls das multiplikative Inverse a^{-1} zu dem Element a existiert, welchen Wert hat es?

Während wir zur Beantwortung der zweiten Frage ein wenig weiter ausholen müssen, lässt sich die erste Frage recht leicht beantworten: Nur wenn m und a zueinander relative Primzahlen sind, wenn also gilt, dass $ggT(a, m) = 1$, so existiert a^{-1}. Welchen Wert a^{-1} dann tatsächlich hat, nun dazu bedarf es des *erweiterten euklidischen Algorithmus* (EEA) als einer Variante des normalen euklidischen Algorithmus. Auch wenn die Berechnung des multiplikativen Inversen sehr wesentlich für das Verständnis der asymmetrischen Kryptografie ist, so wollen wir den EEA hier nicht vorstellen und verweisen stattdessen auf [3].

2.3.3 Chinesischer Restsatz

Der chinesische Restsatz besagt, dass ein System gleichzeitiger Kongruenzen

$$x \equiv a_1 \ (mod\ n_1)$$

$$x \equiv a_2 \ (mod\ n_2)$$

:

$$x \equiv a_q \ (mod\ n_q)$$

mit ganzen Zahlen $n_i > 0$ welche zueinander paarweise teilerfremde natürliche Zahlen $\mathrm{ggT}(n_i, n_j) = 1$ bilden eine eindeutige Lösung modulo $n = n_1 n_2 \ldots n_q$ hat. Einen praktikablen, weil effizienten Ansatz zur Lösung des chinesischen Restsatzes bietet der Gauß-Algorithmus. Der Wert x kann berechnet werden durch Anwendung von

$$\sum_{i=1}^{q} a_i N_i M_i \ mod\ n$$

wobei $N_i = n/n_i$ und $M_i = N_i^{-1}\ mod\ n_i$ [3]. Der chinesische Restsatz und der Gauß-Algorithmus finden in diesem Buch ihre Anwendung, wenn es um die Beschreibung einer Angriffsklasse auf das Bluetooth-Protokoll geht.

2.4 Elementare kryptografische Bausteine

An der einen oder anderen Stelle dieses Buches wird es für den Leser hilfreich sein, bereits ein gewisses Verständnis über den Aufbau und die Fähigkeiten elementarer kryptografischer Bausteine zu haben. Auch wenn es eine Reihe von Standardwerken gibt, allen voran [3] oder [4] in denen dies in einer vorbildlichen und detaillierten Art und Weise aufbereitet ist, so wollen wir dem Leser dennoch in kompakter Form diejenigen Bausteine vermitteln, die für das weitere Verständnis der Inhalte dieses Buches unabdingbar sind. Die hier geleistete kompakte Einführung kann allerdings eine fundierte Auseinandersetzung mit den Grundlagen der angewandten Kryptografie keinesfalls ersetzen, dürfte jedoch ausreichend sein die eine oder andere Passage in diesem Buch besser verstehen zu können.

2.4.1 XOR

Es gibt vermutlich keinen Baustein innerhalb der Kryptografie, der besser verstanden ist als die Modulo-2-Addition. Diese lässt sich in Hardware elegant mittels eines Exklusive-Oder-Gatters, kurz XOR-Gatters, umsetzen und ermöglicht es extrem schnelle Operationen auf Bit-Ebene durchzuführen. So kann ohne wesentliche Zeitverzögerung ein Datenstrom von Bits bit-weise mittels XOR auf einen weiteren Datenstrom von Bits ‚aufaddiert' werden. Ist der eine Bitstrom nun eine Klartextfolge X, die vertraulich über einen unsicheren Kanal zu transportieren ist, und die andere (gleich lange) Bitfolge ein Schlüsselstrom K, so kann auf elegante Weise durch bitweises XOR der einzelnen Bits hieraus senderseitig ein Chiffrestrom C erzeugt werden. Mit dem Wissen um diesen Schlüsselstrom kann durch abermaliges bitweises Aufaddieren auf der Empfängerseite

aus der Chiffre wiederum der ursprüngliche Klartext gewonnen werden. Für eine Klartext-Bitfolge X und eine gleichlange Schlüsselstrom-Bitfolge K gilt daher:

$$X = K \oplus (X \oplus K)$$

Der Ausdruck innerhalb der Klammer ist die Chiffre C welche senderseitig erzeugt und anschließend übertragen wird. Für zwei beispielhafte Bitfolgen für den Klartext $X = 10110010$ und den Schlüsselstrom $K = 10100110$ würde unter Anwendung der Wahrheitswertetabelle für XOR ($\oplus$) (Abb. 2.8) dann der Chiffrestrom $C = 00010100$ übertragen und empfängerseitig durch abermaliges Anwenden zur Entschlüsselung gelangen.

Wenn die unten genannten Bedingungen 1 bis 3 eingehalten sind, lässt sich sogar zeigen, dass dieses XOR-Konstrukt *beweisbare Sicherheit* beziehungsweise vorbehaltlose Sicherheit gegenüber *Ciphertext-only*-Angriffen [8, S. 192] bietet. Den Beweis ersparen wir uns an dieser Stelle und verweisen auf [8] allerdings nicht ohne zu erwähnen, dass solch ein Verschlüsselungsverfahren *One-Time-Pad* genannt wird.

Die drei oben angesprochenen Bedingungen lauten:

Bedingung 1 Zur Erzeugung der Schlüsselstrom-Bitfolge K ist ein *True Random Number Generator* (*TRNG*) erforderlich.

Anmerkung: Das Einhalten dieser Forderung ist nicht so einfach umsetzbar, aber dennoch prinzipiell machbar, auch wenn man für einen echten *TRNG* u. U. jedes Bit per Münzwurf zu generieren hätte sofern man nicht auf andere echte Zufallsquellen zurückgreifen kann. Bei der Umsetzung von ‚Zufall' in der digitalen Welt zeigt sich jedoch sehr schnell, dass wahrer Zufall wie beim Münzwurf nur sehr schwer zu implementieren ist, sodass man oftmals mittels einer Implementierung die Umsetzung von ‚Pseudo'-Zufall (*PRNG*) findet [5]. Mit solch einer Lösung wäre die Umsetzung der Bedingung 1 aber schon nicht mehr erfüllt.

Bedingung 2 Die Zufallsbits sind geheim allein zwischen Sender und Empfänger zu verbreiten

Anmerkung: Auch diese Forderung umzusetzen ist nicht einfach, aber doch prinzipiell einlösbar, selbst wenn man hierzu einen vertrauenswürdigen Kurier mit portablem Datenträger einbeziehen müsste.

A(lice)				B(ob)
$X = 10110010$				
$\oplus\ K = 10100110$				
$C = 00010100$	$\rightarrow$	00010100	$\rightarrow$	$C = 00010100$
				$\oplus\ K = 10100110$
				$X = 10110010$

Abb. 2.8 Beispielhaftes Verschlüsseln und Entschlüsseln mittels XOR ($\oplus$). (Quelle: eigene Darstellung)

Bedingung 3 Die Schlüsselstrombits dürfen nicht mehrfach genutzt werden.

Anmerkung: Diese dritte Forderung ist tatsächlich deutlich schwerer einzulösen, denn sie bedeutet nicht weniger, als dass ein vollständig unabhängiges Schlüsselbit für jedes Klartextbit zu verwenden ist, dass ein Schlüsselstrom immer so lang sein muss wie der zu verschlüsselnde Klartext und dies ohne Abhängigkeiten zwischen dem aktuellen Schlüsselstrombit und allen vorherigen Bits des Schlüsselstroms.

Trotz der obigen Bedingungen ist die Aussage, ein beweisbar sicheres System verstanden und prinzipiell verfügbar zu haben, nicht hoch genug zu bewerten, sagt es doch nichts weniger aus, als dass es einem Angreifer niemals gelingen wird – außer durch zufälliges Raten – einen derartig verschlüsselten Klartext zu entschlüsseln, und zwar auch dann nicht, wenn er hierzu alle auf der Erde und im Universum verfügbare Rechenleistung gleichzeitig verwenden würde, jetzt und in alle Ewigkeit.

Woran liegt das? Nun, es liegt, unter der Voraussetzung, dass die obigen drei Vorbedingungen tatsächlich auch technisch akkurat umgesetzt werden könnten, daran, dass die XOR-Operation (bzw. die Modulo-2-Addition) *perfekt balanciert* ist. Wie den in Abb. 2.9 dargestellten Wahrheitswertetabellen von XOR und der Modulo-2-Addition zu entnehmen ist, liefert deren Ausgabe in Abhängigkeit von den konkret anliegenden Eingaben jeweils zwei 1er- und zwei 0er-Bits. Dies ist beispielsweise bei den logischen Operationen AND, NAND oder OR nicht gegeben. Diese Eigenschaft der perfekten Balanciertheit ist aber entscheidend für die Sicherheit, denn für den Fall, dass ein Klartextbit 0 ist, so ist in Abhängigkeit vom konkreten Schlüsselstrombit das Chiffrat-Bit entweder 0 oder 1. Und zwar mit einer exakt 50 % Chance! Gleiches gilt für den Fall, wenn das Klartextbit 1 ist. Denken Sie daran: Eine Wahrscheinlichkeit von 50 % ob ein Chiffrat-Bit nun ein 0er- oder ein 1er-Bit wird, ist das beste Sicherheitsniveau, welches erreichbar ist. Anders ausgedrückt: Das Wissen das zwei Ereignisse mit jeweils 50 % Wahrscheinlichkeit eintreten können ist eine formalere Beschreibung für *‚pures Raten‘*.

Mit diesen einleitenden Bemerkungen wird vermutlich nachvollziehbarer, warum bei WLAN die Sicherheitsarchitekten des WLAN-Sicherheitsprotokolls der ersten Generation einen so performanten und hinsichtlich seiner Sicherheit derart gut verstandenen und analysierten Baustein wie XOR zur Absicherung der drahtlosen Übertragungsstrecke ihr Vertrauen ausgesprochen haben. Zumal zu Zeiten der Entwicklung von *Wired Equivalent Privacy* (WEP) die Blockchiffre *Data Encryption Standard* (DES) in die Jahre

x	y	$x + y \bmod 2$
0	0	0
0	1	1
1	0	1
1	1	0

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Abb. 2.9 Wahrheitswertetabellen für Modulo-2-Addition und für XOR ($\oplus$)

gekommen war und der *Advanced Encryption Standard* (AES) als deren Nachfolger noch nicht auf der Bühne der Krypto-Verfahren erschienen war.

Allerdings kann man sich nach diesen einleitenden Bemerkungen zu Recht fragen, warum wir dann noch ganze Bücher über die Verwundbarkeit von IT-Systemen schreiben, wenn doch offensichtlich ein beweisbar sicheres und dazu noch extrem leistungsfähiges Verschlüsselungssystem vorhanden ist?

Die Antwort auf diese Frage resultiert aus der technischen Umsetzung der drei oben genannten Vorbedingungen. Wenn man eine praktikable und für die Massennutzung verwendbare Verschlüsselungstechnologie entwerfen möchte, zeigt sich recht schnell, dass das eigentliche Problem in der Erfüllung dieser drei Vorbedingungen liegt. Dies ist auch der Grund, warum praktikable Stromchiffren basierend auf *Linear Feedback Shift Register* (*LFSR*), die ja auch auf XOR beruhen, immer wieder gebrochen werden. Typischerweise ergeben sich je nach Ausprägung und Rückkopplung einer konkreten Anordnung der LFSRs, Zyklenlängen des Schlüsselbitstroms, die dann zwangsläufig zu Wiederholungen der Bitfolgen innerhalb einer Schlüsselstromfolge führen, sodass die Vorbedingung 3 nicht mehr gewahrt ist [4] und die Chiffre damit recht einfach zu brechen ist.

Dennoch hat für die Entwickler des WLAN-Sicherheitsstandards WEP der ersten Generation das allgemein gute Verständnis um die Sicherheitseigenschaften von XOR wohl den Ausschlag gegeben, diesen Baustein auch als elementare Komponente dieses damals neu zu entwickelnden Sicherheitsprotokolls für WLAN einzusetzen. Doch mehr dazu später.

2.4.2 Hashfunktionen

Hash Funktionen sind die einzigen kryptografischen Bausteine die ohne eine Schlüsselkomponente auskommen. Man läuft leicht Gefahr ihre Bedeutung innerhalb einer Sicherheitsarchitektur oder der Verwendung in konkreten Protokollen zu unterschätzen. Doch der Reihe nach. Ein Hashwert x soll als Repräsentant einer Nachricht m dienen. Dabei kann die Nachricht m eine beliebige Länge aufweisen, der Hashwert selbst ist immer von einer festen Länge, die typischerweise deutlich kleiner ist als die Länge der Nachricht. Wir notieren eine Hashfunktion als $x = h(m)$. Die Wahrscheinlichkeit mit der eine vollständig zufällig gewählte Nachricht m einen konkreten Hashwert x erhält beträgt damit $2^{-|x|}$. Mit der obigen Beschreibung dürfte jedoch eine wesentliche Eigenschaft einer Hashfunktion offensichtlich sein: Da der Möglichkeitsraum M aller denkbaren Nachrichten $m \in M$ prinzipiell unendlich groß ist, ist er, egal wie viele Bits die Entwickler einer Hashfunktion für die Länge eines Hashwertes $x \in X$ spendieren, immer deutlich größer als X. Wir notieren dies als $|\mathrm{X}| \ll |M|$, wobei $M \to X$, notiert werden kann als $\{0,1\}^* \to \{0,1\}^{|X|}$. Ohne Beschränkung der Allgemeinheit ist bei dieser Abbildung das Alphabet der binären Symbole $\{0,1\}$ verwendet. Damit gibt es jedoch immer auch Nachrichten $m_1, m_2 \in M$ deren Repräsentanten, also deren Hashwerte x_1 und x_2 identisch sind, sodass $h(m_1) = h(m_2)$. Derartige *Kollisionen* sind also unvermeidbar.

Kryptografische Hashfunktionen $h()$ gehören zu den Einwegfunktionen. Für die Klasse der Einwegunktionen $f : X \rightarrow Y$ gilt, dass $y = f(x)$ leicht zu berechnen ist. Eine Berechnung in die umgekehrte Richtung hingegen ist äußerst schwierig durchzuführen. Das Urbild x zu einem y ist in aller Regel schwer zu bestimmen. Ein wenig formaler lässt sich sagen, dass y in polynomialer Zeit bestimmbar, ein x für ein gegebenes y allerdings in nicht polynomialer Zeit berechenbar ist. Die Abb. 2.10 verdeutlicht das Wesen einer Einwegfunktion.

Neben den zwei elementaren Eigenschaften, die eine Hashfunktion zu erfüllen hat, nämlich der komprimierten Darstellung der Nachricht m durch einen Repräsentanten x sowie der Tatsache, dass x bei einem gegebenen m durch Anwendung von $h()$ leicht und schnell zu berechnen ist, sind noch drei weitere wesentliche Eigenschaften zu nennen:

1. Es ist *berechenbar unmöglich,* bei gegebenem Hashwert x eine Nachricht m zu finden für die $x = h(m)$ gilt.
2. Schwache Kollisionsresistenz: Es ist *berechenbar unmöglich,* eine zweite Nachricht m_2 zu finden, die den gleichen Hashwert liefert wie eine bereits vorhandene Nachricht m_1, sodass $h(m_1) = h(m_2)$ und $m_1 \neq m_2$.
3. Starke Kollisionsresistenz: Es ist *berechenbar unmöglich,* zwei unterschiedliche Nachrichten m_1 und m_2 zu finden, die den gleichen Hashwert bilden, also $h(m_1) = h(m_2)$.

Man beachte, dass im Fall einer starken Kollisionsresistenz ein Angreifer m_1 und m_2 frei wählen kann, während für den Fall, dass die schwache Kollisionsresistenz zu gewährleisten ist, eine Nachricht bereits vorliegt, sodass nur noch die zweite Nachricht frei wählbar ist. Da sich die Anzahl der Freiheitsgrade im Falle der schwachen Kollisionsresistenz gegenüber der starken Kollisionsresistenz halbiert hat, ist es deutlich einfacher für einen Angreifer, die Forderung nach einer starken Kollisionsresistenz einer Hashfunktion zu unterlaufen als der Forderung nach schwacher Kollisionsresistenz.

Der Begriff Nachricht für alle möglichen Elemente aus M ist ein wenig irreführend, da hierin natürlich auch alle Zeichenfolgen enthalten sind, die unter menschlichem Ermessen nicht wirklich als aussagekräftige Nachricht zu deuten sind. Vielmehr handelt

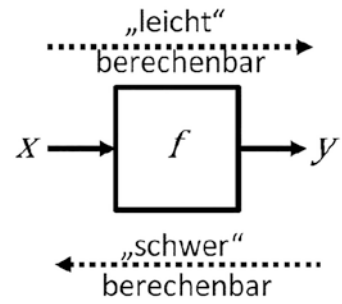

Abb. 2.10 Charakter einer Einwegfunktion $f : X \rightarrow Y$. (Quelle: eigene Darstellung)

es sich bei der Menge M um alle möglichen Zeichenfolgen unter denen sich dann eben auch sinnvolle Nachrichten befinden.

Die Tatsache, dass Kollisionen häufiger auftreten, als uns oft bewusst ist, kann man sich am Geburtstagsparadoxon vergegenwärtigen, das besagt, dass bei 23 Personen in einem Raum die Wahrscheinlichkeit, dass mindestens zwei Personen am gleichen Tage Geburtstag haben schon größer als 50 % ist. Die Beweisidee ist intuitiv einfach: Es soll zunächst die Wahrscheinlichkeit konstruiert werden, dass alle Personen an verschiedenen Tagen Geburtstag haben. Zwei Personen haben also mit der Wahrscheinlichkeit 1-1/365 nicht am gleichen Tag Geburtstag. Um auszudrücken, dass der Geburtstag einer weiteren Person sich ebenfalls von den beiden ersten Geburtstagen unterscheidet, ist die Wahrscheinlichkeit 1-2/365 zu notieren, und so weiter bis alle 23 Fälle berücksichtigt sind, also (1-1/365)(1-2/365)…(1-22/365). Dieser Ausdruck entspricht dem Wert 0,493. Da dieser Wert die Wahrscheinlichkeit angibt mit der keine der im Raum anwesenden Personen mit einer anderen sich im Raum befindenden Person am gleichen Tag Geburtstag hat müssen wir nun $1 - 0{,}493 = 0{,}507$ rechnen um die Wahrscheinlichkeit zu erhalten, dass mindestens zwei Personen an ein und demselben Tage Geburtstag haben. Diese Wahrscheinlichkeit ist mit einem Wert von 50,7 % tatsächlich ein wenig größer als die genannten 50 %. Natürlich nimmt dieser Wert mit jeder weiteren Person zu, die sich in dem Raum befindet.

Das Geburtstagsparadoxon kann als eine Art generischer Angriff auf eine Hashfunktion gedeutet werden, wenn man allein von außen auf die Hashfunktion schaut, ohne sich mit deren innerem Aufbau befasst zu haben. Hierüber kann dann hinsichtlich zu erwartender Kollisionswahrscheinlichkeiten die Mindestlänge für $|x|$ abgeschätzt werden mit der Konsequenz, dass mittlerweile 128-Bit-Hashfunktionen ($2^{64} \approx 10^{19}$) bzw. noch längere Hashwerte als Mindestlänge für $|x|$ genannt werden.

Dass der Entwurf einer guten Hashfunktion welche die obigen geforderten Eigenschaften einlöst und darüber hinaus keinerlei weitreichenden Seiteneffekte bietet, keinesfalls trivial ist und weit über die Betrachtungen des Geburtstagsparadoxons hinausgeht soll an einem Beispiel verdeutlicht werden. Vormals prominente Hashfunktionen wie SHA-1 basieren auf dem Merkle-Damgard-Konstrukt, das im Kern eine Kompressionsfunktion wiederholt anwendet. Hierzu wird zunächst die Eingabenachricht m in Teile gleicher Länge $m_1, m_2, \ldots, m_n$ geteilt. Wie der Abb. 2.11 zu entnehmen ist, geht bei einem Merkle-Damgard-Konstrukt in der i -ten Runde neben dem aktuellen m_i immer auch das Ergebnis der Kompressionsberechnung der vorherigen Runde mit ein. Das Ergebnis der n -ten Runde bildet dann den Hashwert $x = h(m)$ zur Nachricht m. Welchen unerwünschten Nebeneffekt dieses Konstrukt haben kann erörtern wir im nächsten Abschnitt, in dem wir auf Message Authentication Codes eingehen.

Prinzipiell gibt es zwei Varianten eines Entwurfs einer Hashfunktion. Zum einen kann eine Hashfunktion dediziert entwickelt werden wie über das Merkle-Damgard-Konstrukt, zum anderen können andere Komponenten genutzt werden, wie der Einsatz von Blockchiffren beispielsweise unter Verwendung des *Advanced Encryption Standard* (AES).

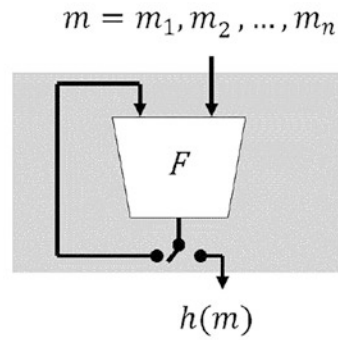

Abb. 2.11 Aufbau einer Hashfunktion nach Merkle-Damgard-Konstruktion. (Quelle: eigene Darstellung in Anlehnung an [4])

In den letzten Jahren sind eine Reihe von Hashfunktionen ‚gekippt', sie gelten also als nicht mehr sicher. So sollten MD5 und SHA-1 nicht mehr verwendet werden, für die SHA-2-Familie bestehend aus SHA-224, SHA-256, SHA-384 und SHA-512 mit Hashwerten der Längen 224, 256, 384 sowie 512 Bit sind offiziell noch keine Kollisionen bekannt. Dennoch wurde 2015 vom *National Institute of Standards and Technology* (NIST) mit SHA3 eine neue Familie von Hashfunktionen standardisiert: SHA-3-224, SHA-3-256, SHA-3-384 sowie SHA-3-512.

2.4.3 Message Authentication Codes

Message Authentication Codes, kurz MACs werden als kryptografische Prüfsumme unter Verwendung eines gemeinsamen symmetrischen Schlüssels k eingesetzt. Eine beliebig lange Nachricht m ist die Eingabe und heraus kommt eine Prüfsumme fester Länge x, also $x = MAC_k(m)$. Da eine Prüfsumme nur gemeinsam mit der Nachricht auswertbar ist, wird typischerweise (m, x) an den Empfänger übertragen (Abb. 2.12).

Bei einer erfolgreichen Verifikation der Prüfsumme beim Empfänger $(x' ? = x)$ hat sich dieser über die Integrität und die Authentizität der Nachricht vergewissert: Unter der Bedingung dass k nur Alice und Bob bekannt ist und keiner dritten Partei, weiß Bob, dass die Nachricht von Alice stammt. Bob kann mittels des MAC ebenfalls erkennen ob

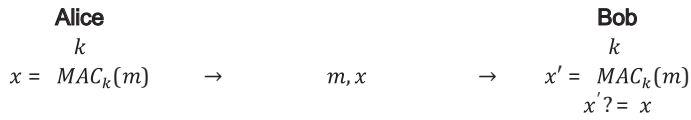

Abb. 2.12 Verwendung eines MAC zur Überprüfung der Integrität und Authentizität einer Nachricht. (Quelle: eigene Darstellung in Anlehnung an [4])

eine Manipulation an der Nachricht oder der Prüfsumme auf der Übertragungsstrecke vorgenommen worden ist.

Eine sehr verbreitete Umsetzung eines Message Authentication Codes ist der HMAC. Dieser besteht aus einer Hashfunktion und einer geschickten Anordnung des Schlüssels. Der HMAC wurde im Jahre 1996 von den Kryptologen Bellare, Canetti und Krawczyk vorgeschlagen und behebt insbesondere die Sicherheitsmängel von naiven Lösungen wie der des Secret Prefix MAC oder des Secret Suffix MAC. Hierzu besteht der HMAC aus einem inneren und einem äußeren Hash, sowie der zweimaligen Verwendung des Schlüssels k:

$$x = HMAC_k(m) = h[(k \oplus opad)||h[(k \oplus ipad)||m]]$$

Neben der Nachricht m und dem Schlüssel k werden noch Padding-Konstanten für ein inneres Padding und ein äußeres Padding *ipad* und *opad* genutzt und XOR ($\oplus$) darauf angewendet. Die genauen Bitfolgen dieser beiden Padding-Konstanten erspare ich uns jedoch.

Bellare, Canetti und Krawczyk haben gezeigt, dass der HMAC *beweisbar sicher* ist. Das heißt, ein Angreifer kann den HMAC nur dann brechen, wenn er die eingesetzte Hashfunktion $h()$ bricht. Die Verwendung des HMAC zur Überprüfung der Integrität und Authentizität einer Nachricht ist in Abb. 2.13 dargestellt.

Manch Leser mag sich nun fragen warum man nicht einen MAC in solch komplexer Bauart konstruiert, wie sie der HMAC aufweist. So könnte beispielsweise $x = MAC_k(m)$ deutlich einfacher konstruiert werden durch Voranstellen des Schlüssels an die Nachricht: $x = h(k||m)$. Als MAC dient also die gehashte Nachricht m, der der Schlüssel k vorangestellt ist. Basiert nun $h()$ auf einem Merkle-Damgard-Konstrukt, so können wir die zu hashende Nachricht m in der Form $m = (m_1, m_2, \ldots, m_n)$ notieren. Dies führt zu $x = MAC_k(m) = h(k||m_1, m_2, \ldots, m_n)$. Doch dann wäre Mallory in der Lage seine eigene Nachricht der Bauart $m_1, m_2, \ldots, m_n, m_{n+1}$ mittels dieses MAC zu sichern und von Bob authentifizieren zu lassen, ohne selbst im Besitz des gemeinsamen Schlüssels von Alice und Bob zu sein. Er fügt also an die abgefangene Nachricht m noch den Zusatz m_{n+1} und sendet die Nachricht $m_M = m_1, m_2, \ldots, m_n, m_{n+1}$ an Bob. Gleichzeitig hält er die Prüfsumme x zurück und berechnet seinen eigene Prüfsumme $x_M = h(m||m_{n+1})$ ohne, dass er hierzu den Schlüssel k benötigen würde. Die Abfolge eines solchen Angriffs ist in der Abb. 2.14 dargestellt.

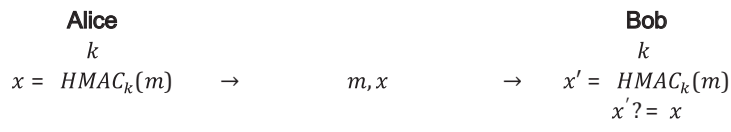

Abb. 2.13 Verwendung eines HMAC zur Überprüfung der Integrität und Authentizität einer Nachricht. (Quelle: eigene Darstellung)

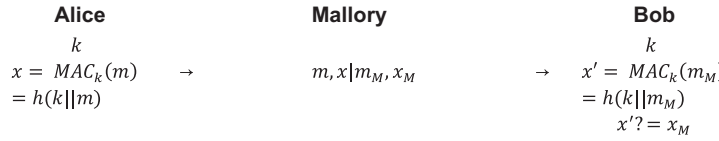

Abb. 2.14 Verwundbarkeit eines naiven MAC-Konstruktes mit MitM-Mallory. (Quelle: eigene Darstellung in Anlehnung an [4])

Der oben geschilderte Angriff ist aufgrund der Bauart des MAC, bei der der Schlüssel vorangestellt wird, in der Literatur bekannt als Angriff auf einen Secret Prefix MAC. Ähnliche Angriffe lassen sich auch auf MACS konstruieren, bei denen der Schlüssel an das Ende der Nachricht gestellt ist (Secret Suffix MAC) und die Hashfunktion auf diese Daten angewandt wird. Da bei Verwendung eines HMAC derartige Angriffe ausgeschlossen sind, sollte man sich im Zweifelsfall bei der Wahl eines MAC immer eines HMAC bedienen.

2.4.4 Digitale Signaturen

Mit dem Aufkommen der asymmetrischen Kryptografie oder Public-Key-Kryptografie Mitte der 70er-Jahre hat ein Ansatz Eingang in die Kryptografie gefunden mit dem das Problem der Schlüsselverteilung substanziell erleichtert wurde. Wie auch aus den Abbildungen zur Verwendung von MACs im vorigen Abschnitt hervorgeht, hat der symmetrische Schlüssel k den Parteien Alice und Bob bereits vorzuliegen bevor ein symmetrischer Ansatz wie eine MAC-Berechnung oder eine symmetrische Verschlüsselung durchführbar ist. Bei Verwendung symmetrischer Kryptografie müssen die beiden Parteien sich also vorab über einen gesicherten Kanal auf den Schlüssel k verständigt haben. Der Terminus ‚gesicherter Kanal' soll darauf hinweisen, dass keine dritte Partei bei diesem Vorgang die Möglichkeit erlangen darf den Schlüssel zu erfahren oder aber diesen unerkannt zu ändern.

Seit dem Aufkommen asymmetrischer kryptografischer Verfahren haben wir es nun mit einem gänzlich anderen Konzept zu tun. Zum Einsatz kommt ein Schlüsselpaar. Der öffentliche Schlüssel k_{pub} birgt kein Geheimnis und kann daher jeder denkbaren Partei auf öffentlichem Weg mitgeteilt werden solange die Authentizität des Schlüssels bei seiner Übertragung gewährleistet ist, der Empfänger also den Sender zweifelsfrei zuordnen kann und in der Lage ist, eventuelle Manipulationen des Schlüssels seiner Übertragung zu erkennen.

Anders steht es um den privaten Schlüssel k_{pr}. Dieser darf allein seinem Besitzer bekannt sein und muss daher sorgfältig von diesem abgelegt und verwahrt werden. Eine Signatur s wird nun gebildet auf einer Nachricht m und dem privaten Schlüssel k_{pr} unter Anwendung eines Signaturerfahrens $Sig()$. Wir notieren dies mit $s = Sig_{kpr}(m)$. Jede Partei die den korrespondierenden Schlüssel k_{pub} kennt, ist nun in der Lage, die so erstellte

Signatur s zu prüfen. Durch Anwendung einer Funktion $true/false = Ver_{kpub}\ (s, m)$ kann nun jede Partei, welche den öffentlichen Schlüssel kennt, überprüfen, ob die Signatur s zur Nachricht m passt. Dies ist der Fall, wenn $true = Ver_{kpub}\ (s, m)$. Ansonsten ist davon auszugehen, dass es entweder eine Manipulation der Nachricht, der Signatur oder aber beider Bestandteile gegeben hat. Ebenso wie bei einer Prüfung mit einem MAC bieten digitale Signaturen die Prüfung der Authentizität und der Integrität einer Nachricht. Darüber hinaus besitzen digitale Signaturen jedoch noch eine weitere wertvolle Eigenschaft: die Nicht-Abstreitbarkeit. Das Schutzziel der Nicht-Abstreitbarkeit bietet ein MAC beispielsweise nicht. Was hat es damit auf sich? Nun, der Sender einer signierten Nachricht kann nicht abstreiten, der Urheber dieser Nachricht zu sein, sofern sein privater Schlüssel nicht in fremde Hände gelangt ist.

Da die Verwendung asymmetrischer Kryptografie deutlich rechenintensiver ist als die Verwendung symmetrischer Verfahren, und der Rechenaufwand mit der Länge der Nachricht m anwächst, wird zunächst der Hashwert der Nachricht m gebildet. Signiert wird damit nicht die Nachricht selbst, sondern ihr Hashwert. Dies ist in der Abb. 2.15 dargestellt.

Auch wenn sich mit der Zeit einige weitere technische Umsetzungen für digitale Signaturen etabliert haben, so begnügen wir uns an dieser Stelle mit dem ersten bekannt gewordenen Verfahren, welches die obigen geforderten Eigenschaften tatsächlich praktikabel umgesetzt hat. Das RSA-Verfahren, benannt nach den ersten Buchstaben im Nachnamen seiner Erfinder Rivest, Shamir und Adleman, kann sowohl zum Verschlüsseln als auch zum Signieren einer Nachricht genutzt werden. Wir stellen es hier allein für die Signaturbildung vor da es für diesen Zweck vermutlich weitaus häufiger Verwendung findet als zur Verschlüsselung.

Der öffentliche Schlüssel k_{pub} besteht aus dem Tupel der Zahlen (n, e). Der private Schlüssel k_{pr} ist eine Zahl d, sodass sich notieren lässt:

$$k_{pub} = (n, e)$$

$$k_{pr} = d$$

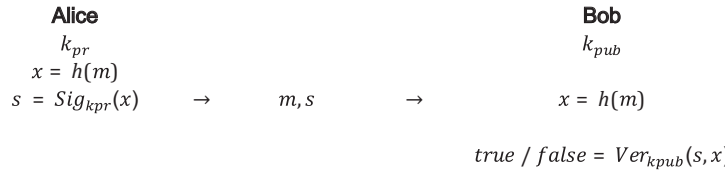

Abb. 2.15 Signaturbildung unter Verwendung einer Hashfunktion durch Alice und deren Prüfung durch Bob. (Quelle: eigene Darstellung in Anlehnung an [4])

Unter RSA erfolgt die Signaturbildung $s = Sig_{kpr}(x)$ einer Zahl x nun wie folgt:

$$s = x^d \ mod \ n$$

Die Prüfung der Signatur über $Ver_{kpub}(s, x)$ geschieht bei RSA wie folgt:

$$x = s^e \ mod \ n$$

Wie genau das Schlüsselpaar $k_{pub} = (n, e)$ und $k_{pr} = d$ zum einen passgenau und zum anderen mit einem begründbar guten Sicherheitsniveau generiert werden können wollen wir nun skizzieren.

Allerdings sei angemerkt, dass für ein fundiertes Verständnis ein gewisses zahlentheoretisches Vorwissen erforderlich ist, das sicherlich den Rahmen dieses Buches übersteigt. Wir verweisen hierzu auf [3]. Daher ist hier nur das ‚Kochrezept' angegeben, ohne eine tiefergehende Betrachtung zu den Eigenschaften bzw. einer vorteilhaften Wahl der einzelnen Parameter vorzunehmen:

- Zwei große Primzahlen p und q sind zu wählen. Diese sollten ungefähr die gleiche Größenordnung haben
- Berechne $n = pq$ sowie $\phi = (p-1)(q-1)$.
- Wähle eine zufällige Zahl e (der öffentliche Exponent) aus $1 < e < \phi$. Dabei muss $ggT(e, \phi) = 1$ erfüllt sein.
- Verwende zur Berechnung von d (der private Exponent) mit $1 < d < \phi$ den *erweiterten euklidischen Algorithmus,* sodass $ed \equiv 1 \ (mod \ \phi)$

Nach der RSA-Schlüsselgenerierung sind (n, e) als öffentlicher Schlüssel und d als privater Schlüssel zu verwenden. Man muss sich darüber im Klaren sein: Einzelne Schritte der RSA-Schlüsselgenerierung sind mitnichten trivial. So ist es unabdingbar, große und geeignete Primzahlen p und q zu finden. Die Berechnung des öffentlichen Exponenten e und des privaten Exponenten d erfolgt unter Anwendung des erweiterten euklidischen Algorithmus und führt zur Lösung der Gleichung

$$ggT(\phi, e) = s\phi + te$$

mit den beiden ganzzahligen Koeffizienten s und t. Hierbei ist $-t$ das Inverse des öffentlichen Exponenten e. Der private Schlüssel d ist dann $d = t \ mod \ \phi$. Diejenigen die sich mit der Arbeitsweise des erweiterten euklidischen Algorithmus vertraut machen möchten oder aber tiefere Einblicke in Eulers Phi-Funktion erhalten wollen, seien neben anderen guten Quellen hierzu insbesondere auf [3] verwiesen.

Abschließend sei noch darauf hingewiesen, dass das eingangs beschriebene Konstrukt der Signatur einer Nachricht auf seinem Hashwert und nicht auf der Nachricht selbst nicht ganz unproblematisch ist. Denn findet Mallory zu einer Nachricht m_1, zu der bereits eine Signatur s vorliegt, eine zweite kollidierende Nachricht m_2, mit der $x = h(m_1) = h(m_2)$ gilt, so kann er die Signatur s verwenden und Bob die Signatur s

gemeinsam mit m_2 zusenden. Die Prüfung wäre dann erfolgreich, $true = Ver_{kpub}\ (x, s)$, ohne dass Mallory jemals in Besitz des privaten Schüssels von Alice gekommen wäre. Bob, der Empfänger dieser so signierten Nachricht, würde jedoch davon ausgehen, dass nur Alice diese Nachricht signiert haben konnte. Oder: Mallory hätte fälschlicherweise eine Signatur von Alice für seine Nachricht m_2 verwendet, ohne im Falle von RSA beispielsweise im Besitz des privaten Schlüssels $k_{pr} = d$ von Alice zu sein.

Neben dem hier eingeführten Schulbuch-RSA gibt es insbesondere aufgrund seiner multiplikativen Eigenschaften noch eine Reihe von weiterführenden Aspekten, um RSA tatsächlich in der Praxis sicher verwenden zu können und gegen solche sogenannten ‚Dehnbarkeits'-Angriffe zu härten. Doch auch hier sei auf die weiterführende Literatur zum Thema verwiesen.

2.4.5 Verschlüsselung

Möchte man das Schutzziel der Vertraulichkeit in der digitalen Welt umsetzen, so wird hierzu die Technik der Verschlüsselung verwendet. Alice überführt also vor dem Versenden den zu übermittelnden Klartext *a* in ein Chiffrat c indem sie hierzu einen Schlüssel verwendet. Ein Chiffrat ist dabei eine Folge von Zeichen mittels derer ein Angreifer idealerweise keinerlei Rückschlüsse auf den ursprünglichen Klartext *a* erhalten kann. Je nach konkreter Ausgestaltung solch eines Verschlüsselungsverfahrens kann die Länge eines Chiffrats von dem Klartext abweichen. Enthält der Klartext identische Passagen, werden ernstzunehmende Verschlüsselungsverfahren so konzipiert sein, dass dies durch bloßes Draufschauen auf das Chiffrat nicht ersichtlich ist. So sind beim Entwurf eines Verschlüsselungsverfahrens noch eine ganze Reihe weiterer Aspekte zu berücksichtigen, um eine schwer zu brechende Chiffre zu entwerfen.

In den nachfolgenden Kapiteln notieren wir die Verschlüsselung mit $E()$immer dann, wenn es zunächst einmal unerheblich ist, welches konkrete Verfahren zum Einsatz kommt. Das Verfahren zum Entschlüsseln notieren wir als $D()$. Wird sowohl die Verschlüsselung als auch die Entschlüsselung mit demselben Schlüssel *k* durchgeführt, dann sprechen wir von einem *symmetrischen Verschlüsselungsverfahren.* Wir können für alle Klartexte *a* und alle symmetrischen Schlüssel *k* notieren:

$$a = D_k(E_k(\mathrm{a}))$$

Alice und Bob können also ein Verfahren zur symmetrischen Verschlüsselung anwenden um auch in Gegenwart von Eve die Nachricht *a* vertraulich zu übertragen sofern sie sich vorab vertraulich über den gemeinsamen Schlüssel *k* verständigt haben (Abb. 2.16.).

Solche Verschlüsselungsverfahren zu entwerfen ist nicht einfach und man sollte dies unbedingt erfahrenen Kryptografen überlassen. Und auch dann kann noch eine ganze Menge schiefgehen wie wir in den nachfolgenden Kapiteln noch erörtern werden.

Zum jetzigen Zeitpunkt unserer Betrachtungen soll es ausreichend sein, festzuhalten, dass es bei den symmetrischen Verschlüsselungsverfahren zwei Arten gibt: die *Stromchiffren* und die *Blockchiffren.* Während die Stromchiffren die Verschlüsselung eines Klartextes bitweise vornehmen – jedes Bit des Klartextes wird einzeln verschlüsselt –, werden bei Blockchiffren jeweils Blöcke verschiedener Länge, beispielsweise mit einer Blocklänge von 128 Bit, gemeinsam zu einem Chiffrat-Block verschlüsselt. Ist der zu verschlüsselnde Klartext dann größer als der über das Verschlüsselungsverfahren vorgesehene Block, so müssen mehrere Blöcke verschlüsselt werden. Allerdings müssen diese einzeln erzeugten Chiffrat-Blöcke dann noch in einer geeigneten Art und Weise miteinander verkettet werden. Hierzu werden sogenannte *Modes-of-operation* verwendet, die selbst wieder einen nicht unerheblichen Einfluss auf die Sicherheit des erzeugten Chiffrats c haben, das sich über mehrere Chiffrat-Blöcke erstreckt. Denn würde diese Verkettung nicht vorliegen, dann könnte ein Angreifer Mallory beispielsweise Chiffrat-Blöcke umordnen oder verwerfen, ohne dass Bob dies bei der Entschlüsselung $a = D_k(\mathrm{c})$ auffallen würde.

Ein Baustein zur symmetrischen Stromverschlüsselung ist Ihnen bereits bekannt: In Abschn. 2.4.1 haben wir den Baustein XOR besprochen und seine Verwendung innerhalb der Kryptografie bereits ausgiebig gewürdigt. Verwendet man wie gesehen XOR zur bitweisen Verschlüsselung so sind $E()$ und $D()$ identisch. Allerdings möchte ich noch einmal darauf hinweisen, dass der in Abb. 2.8 verwendete Schlüsselstrom K nicht dem Schlüssel k aus der Abb. 2.16 entspricht. Denn ein Schlüsselstrom K hat immer die Länge des Klartextes selbst wenn es sich hierbei um eine sehr lange Nachricht handeln sollte. Der Schlüssel k dagegen ist bei Blockchiffren oftmals in der Größenordnung der Blocklänge. Bei Stromchiffren geht k dagegen in die Erzeugung des Schlüsselstroms K ein. Im Normalfall ist der Schlüsselstrom also deutlich länger als der Schlüssel.

Die Funktionen $E()$ und $D()$ sind bei symmetrischen Blockchiffren wie dem Advanced Encryption Standard (AES) oder 3DES ähnlich, jedoch nicht vollständig identisch. Der AES ist seit seiner formalen Freigabe durch die NIST im Jahre 2001 die bewährte symmetrische Blockchiffre, die im Zweifel, bis auf Weiteres, anderen symmetrischen Verfahren vorzuziehen ist. Seine Blocklänge beträgt 128-Bit. Die Schlüssellänge ist wählbar zwischen 128-Bit, 192-Bit und 256-Bit. Im Verlaufe dieses Buches werden wir allerdings erkennen, dass in mobilen Systemen noch eine Reihe weiterer symmetrischer

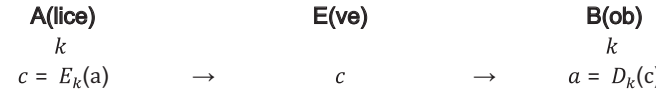

Abb. 2.16 Darstellung einer verschlüsselten Kommunikation mittels *symmetrischer* Verschlüsselung in Gegenwart eines passiven Angreifers (E)ve. (Quelle: eigene Darstellung)

Verschlüsselungsverfahren zur Anwendung gelangen, beispielsweise weil der AES zum Einführungszeitpunkt eines Sicherheitsprotokolls zur drahtlosen Übertragung noch gar nicht zur Verfügung stand. Wir werden aber auch feststellen, dass allein die Verwendung des AES noch keine Garantie für ein schwer zu brechendes Gesamtsystem bietet.

Wir sprechen hingegen von einem *asymmetrischen Verschlüsselungsverfahren,* wenn der Klartext a unter Verwendung asymmetrischer Kryptografie verschlüsselt wurde. Dabei wird der öffentliche Schlüssel k_{pub} zum Verschlüsseln verwendet und der private Schlüssel k_{pr} zum Entschlüsseln:

$$a = D_{kpr}(E_{kpub}(\mathrm{a}))$$

In der Abb. 2.17 ist die verschlüsselte Kommunikation mittels *asymmetrischer* Verschlüsslung in Gegenwart eines passiven Angreifers E(ve) dargestellt. Möchte Alice eine Nachricht für Bob verschlüsseln, so nutzt Alice hierfür den öffentlichen Schlüssel $k_{B.kpub}$ von Bob. Erhält Bob das Chiffrat c von Alice, so verwendet Bob seinen eigenen privaten Schlüssel $k_{B.kpr}$, um die Nachricht zu entschlüsseln.

Bei allen Vorteilen hinsichtlich der Schlüsselverbreitung, welche die asymmetrische Verschlüsselung gegenüber der symmetrischen Verschlüsselung bietet, hat sie jedoch einen großen Nachteil: Das Verschlüsseln ist je nach verwendetem Verfahren zwischen Faktor 10^2 und Faktor 10^3 rechenintensiver. Dies ist auch der Grund, warum man mittels asymmetrischer Kryptografie eher kleinere Klartexte verschlüsselt. Gerne wird der in Abb. 2.17 dargestellte Vorgang daher zur sicheren Bekanntmachung eines symmetrischen Schlüssels k zwischen Alice und Bob verwendet. Alice berechnet also $c = E_{kB.pub}(k)$ sodass Bob k erhält, indem er $k = D_{kB.pr}(\mathrm{c})$ berechnet. Ist dies geschehen, können die beiden Parteien anschließend die eigentliche Nachricht a unter Verwendung von k und eines symmetrischen Verfahrens wie AES verschlüsseln.

Ein asymmetrisches Verfahren zur Verschlüsselung ist Ihnen bereits bekannt: RSA. Allerdings haben Sie es bisher nur zum digitalen Signieren von Nachrichten kennen gelernt (siehe Abschn. 2.4.4). Es kann aber auch zur Verschlüsselung kleinerer Nachrichten verwendet werden, indem man den öffentlichen Schlüssel $k_{pub} = (n, e)$ und den privaten Schlüssel $k_{pr} = d$ im Gegensatz zur digitalen Signaturbildung einer Nachricht d in umgekehrter Reihenfolge anwendet. Die Verschlüsselung der Nachricht a mit RSA lautet demnach:

$$c = E_{kpub}(a) = a^e \ mod \ n$$

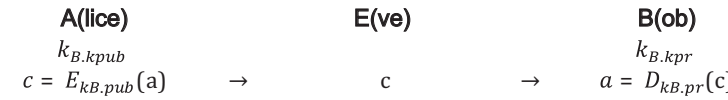

Abb. 2.17 Darstellung einer verschlüsselten Kommunikation mittels *asymmetrischer* Verschlüsselung in Gegenwart eines passiven Angreifers (E)ve. (Quelle: eigene Darstellung)

und die Entschlüsselung:

$$a = D_{kpr}(c) = c^d \ mod \ n$$

Für weitere Details möchte ich Sie wiederum auf [3] verweisen.

2.4.6 Schlüsselübereinkunft

Die Basisversion der *Diffie-Hellman-Schlüsselübereinkunft* stammt aus dem Jahre 1976. Whitfield Diffie und Martin Hellman publizierten 1976 das Protokoll in einem Aufsatz mit dem Titel „New Directions in Cryptography" [2] und erhielten im Juni 2015 hierfür den ACM Turing Award. In der Laudatio zu diesem Festakt sagte ACM-Präsident Alexander Wolf:

> „1976 sahen Diffie und Hellman eine Zukunft voraus, in der Menschen tagtäglich über elektronische Netzwerke kommunizieren und darin verletzlich waren, dass ihre Kommunikation gestohlen oder verändert wird. Nun sehen wir 40 Jahre später, dass ihre Überlegungen bemerkenswert vorausschauend waren."

Hellman gab sogar in seinen Memoiren an, einen Mordanschlag zu fürchten, da man ihm vorwarf, mit der Veröffentlichung die nationale Sicherheit der USA zu untergraben.

Doch was genau ist der wissenschaftliche und praktische Wert der Arbeit dieser beiden Kryptografen: Ziel dieses Protokolls zur Schlüsselübereinkunft ist es, dass sich zwei Parteien, Alice und Bob, auf einen gemeinsamen geheimen symmetrischen Schlüssel verständigen können und das obwohl sie einzig über einen unsicheren Kommunikationskanal verfügen. Hierzu verwenden sie die asymmetrische Kryptografie welche wenige Jahre zuvor durch den für das GCHQ arbeitenden Briten James Ellis entdeckt und bewiesen wurde, aber aus Gründen der Staatssicherheit unter Verschluss blieb.

In einer anfänglichen Phase einigen sich die beiden Parteien auf eine geeignete Primzahl p sowie ein Generatorelement α für $\mathbb{Z}_p^*$ mit $2 \leq \alpha \leq p-2$. Beide Werte bilden kein Geheimnis und können daher öffentlich beispielsweise via besagten unsicheren Kanal übertragen werden. Nun wählt Alice ein zufälliges Geheimnis x aus dem Wertebereich $1 \leq x \leq p-2$ aus und Bob wählt seinerseits ein zufälliges Geheimnis y aus dem Wertebereich $1 \leq y \leq p-2$ aus. Danach sendet Alice die Nachricht $\alpha^x mod\ p$ an Bob und Bob sendet die Nachricht $\alpha^y mod\ p$ an Alice (Abb. 2.18). Mit dem Erhalt dieser Nachrichten sind nun beide Parteien in der Lage, den gemeinsamen Sitzungsschlüssel k unabhängig voneinander zu berechnen. Hierzu berechnet Alice $k = (\alpha^y)^x mod\ p$ und Bob berechnet $k = (\alpha^x)^y mod\ p$.

Die *Korrektheit* des Protokolls lässt sich überraschend einfach über die Kongruenzbeziehung aufzeigen. So gilt $k = (\alpha^y)^x \equiv \alpha^{xy} mod\ p$. Weiter gilt $k = (\alpha^x)^y \equiv \alpha^{xy} mod\ p$.

Alice				Bob
α, p	$\rightarrow$	α, p	$\rightarrow$	α, p
x	$\rightarrow$	$\alpha^x mod\ p$	$\rightarrow$	
	$\leftarrow$	$\alpha^y mod\ p$	$\leftarrow$	y
$k = (\alpha^y)^x mod\ p$				$k = (\alpha^x)^y mod\ p$

Abb. 2.18 Abfolge der Diffie-Hellman-Schlüsselübereinkunft. (Quelle: eigene Darstellung)

Doch was wissen wir über die *Sicherheit* der Diffie-Hellman-Schlüsselübereinkunft? Hierbei ist zwischen einem passiven Angreifer Eve und einem aktiven Angreifer Mallory zu unterscheiden. Beide haben das erklärte Ziel den Schlüssel k zu erlangen.

Beginnen wir mit Eve: Eve erhält durch Mithören α, p, $\alpha^x mod\ p$, sowie $\alpha^y mod\ p$. Der *angenommen* einzige Weg, an den Schlüssel k zu gelangen, besteht für Eve ausschließlich darin, das Problem des diskreten Logarithmus (DLP) zu lösen. Dies bedeutet, dass Eve innerhalb einer multiplikativen Gruppe mit Primzahl p ein Element x finden muss, sodass gilt:

$$x \equiv log_\alpha \alpha^x mod\ p$$

Das DLP wird für genügend große Zahlen als sehr schwer lösbar eingestuft, sodass es Eve ohne weitere Informationen nach mathematischem Verständnis kaum möglich sein dürfte, auf den Schlüssel k zu schließen. Wem diese Erläuterung aus verständlichen Gründen jedoch formal nicht ausreicht, der sei beispielsweise auf Kap. 3 des Buches ‚Cryptography Made Simple' von Nigel Smart [6] verwiesen.

Was aber kann Mallory ausrichten? Geht man von einem *aktiven* MitM-Angreifer aus, der Nachrichten auf der Übertragungsstrecke zwischen Alice und Bob sowie in umgekehrter Richtung nicht nur mithört, sondern auch nach seinen Belangen manipuliert oder verwirft, so ist es bekanntes Lehrbuchwissen, dass die Basisversion der Diffie-Hellman-Schlüsselübereinkunft unsicher gegenüber einem derartigen Angreifer Mallory ist.

Gegen einen aktiven MitM-Angreifer ist es bekanntermaßen erforderlich, dass sich die Parteien Alice und Bob auch gegenseitig authentifizieren. Denn ist dies nicht der Fall so kann Mallory Nachrichten verändern. Er könnte, wie in Abb. 2.19 gezeigt, die Nachricht $\alpha^x mod\ p$ von Alice ändern in $\alpha^z mod\ p$ und die Nachricht $\alpha^y mod\ p$ von Bob ändern in $\alpha^z mod\ p$, indem er anstelle der Geheimnisse x und y sein eigenes Geheimnis z einsetzt.

Alice		Mallory		Bob
α, p	$\rightarrow$	α, p	$\rightarrow$	α, p
x	$\rightarrow$	$\alpha^x mod\ p \mid \alpha^z mod\ p$	$\rightarrow$	
	$\leftarrow$	$\alpha^z mod\ p \mid \alpha^y mod\ p$	$\leftarrow$	y
$k_{AM} = (\alpha^z)^x mod\ p$		$k_{AM} = (\alpha^z)^x mod\ p, k_{MB} = (\alpha^z)^y mod\ p$		$k_{MB} = (\alpha^z)^y mod\ p$

Abb. 2.19 Abfolge der Diffie-Hellman-Schlüsselübereinkunft mit MitM-Mallory. (Quelle: eigene Darstellung)

Dies führt dazu, dass Mallory mit Alice den gemeinsamen Schlüssel k_{AM} ausgehandelt hat und mit Bob den gemeinsamen Schlüssel k_{MB}. Sowohl Alice als auch Bob würden zukünftig diese Schlüssel zum Verschlüsseln von Nachrichten verwenden, die allein zur Kommunikation zwischen Alice und Bob gedacht sind. Schaltet sich Mallory auch dann noch als Proxy in die Kommunikation ein, so denken Alice und Bob, dass sie weiterhin geheim kommunizieren, und erkennen nicht, dass jegliche verschlüsselte Kommunikation von Mallory entschlüsselt und damit mitgelesen werden kann.

2.4.7 Zertifikate

Zertifikate sind eine zwingend erforderliche Teilkomponente, damit asymmetrische Kryptografie überhaupt breit einsetzbar ist. Und selbst dann, so werden wir später erkennen, kann noch eine ganze Menge schiefgehen. Doch der Reihe nach: In Abschn. 2.4.4 haben wir erfahren, dass der öffentliche Schlüssel k_{pub} zu einem privaten Schlüssel k_{pr} selber kein Geheimnis birgt und daher auf öffentlichem Weg der Gegenstelle Bob mitgeteilt werden kann. Dabei sollte ein öffentlicher Schlüssel immer auch die Information enthalten, wem er gehört. Wir können dies in Anlehnung an [4] der Einfachheit halber notieren als $K_A = (ID_A, k_{pub,A})$. Hierbei ist ID_A der Identifikator von Alice und $k_{pub,A}$ sein öffentlicher Schlüssel. Allerdings könnte ein Angreifer Mallory nun eine Änderung vornehmen, sodass Bob $K_A = (ID_A, k_{pub,M})$ erhält. Da dies im Wesentlichen in Hexadezimaldarstellung ohne erkennbare Struktur erfolgt, wird ein ungeübter Nutzer Bob diese Manipulation nicht erkennen können. Anhand dieses Beispiels wird deutlich, dass, die asymmetrische Kryptografie zwar keinen vertraulichen Kanal zum Übermitteln öffentlicher Schlüssel benötigt, aber einen authentifizierten Kanal. Genau dies ist der Grund warum es Zertifikate gibt. Denn diese bieten ein technisches Mittel den korrespondierenden öffentlichen Schlüssel zu einem privaten Schlüssel an die Identität des Eigentümers zu binden. In seiner technischen Umsetzung ist ein Zertifikat dann jedoch wiederum nichts anderes als die Verwendung einer digitalen Signatur. Dies können wir in vereinfachter Darstellung notieren als $Cert_{CA} = (ID_A, k_{pub,A}, s)$. Hierbei geht in das Zertifikat eine digitale Signatur s ein unter der Nutzung eines privaten Schlüssels k_{prCA} einer vertrauenswürdigen Instanz, der Zertifikatsstelle (engl. *Certificate Authority*). Die Signatur ist hierbei $s = Sig_{kprCA}\ (h(ID_A, k_{pub}))$. Die Ausgabe solch digitaler Zertifikate zur zweifelsfreien Bindung der Identität einer Person oder Entität an ihren öffentlichen Schlüssel wird nicht nur von staatlichen Organisationen, sondern auch von privaten Unternehmen wie Google oder Symantec durchgeführt, wobei innerhalb einer *Public-Key-Infrastruktur (PKI)* besonders qualifizierte Zertifikatsstellen auch anderen Zertifikatsstellen Zertifikate ausstellen dürfen. Es ist schwierig, da noch den Überblick zu behalten, wer in diesem Vertrauensrückgrat, dem sogenannten *Network of Trust,* für eine sichere weltweite digitale Kommunikation tatsächlich vertrauenswürdig ist und welche Zertifikate auf unseren Geräten abgelegt sind, die von nicht vertrauenswürdigen Zertifikatsstellen ausgestellt worden sind. Dies kann beispielsweise dazu führen, dass

eine mit *Transport Layer Security* (TLS) gesicherte verschlüsselte Verbindung jederzeit von einem Aussteller nicht vertrauenswürdiger Zertifikate aufgebrochen und mitgelesen werden kann. Als weitere Fallstricke sind die Gültigkeitsdauer von Zertifikaten zu nennen sowie das Ausstellen sogenannter selbstsignierter Zertifikate. Denn auch wenn Zertifikate neben anderen Feldern über ein Feld verfügen, die kennzeichnet, bis wann ein Zertifikat längstens gültig ist, so kann auch die Verwendung sogenannter Zertifikatssperrlisten (engl. *Certificate Revocation List* [CRL]) nicht gewährleisten, dass Sicherheitsprüfungen auf Ihren digitalen Geräten nicht doch zumindest für einen gewissen Zeitraum auf der Basis veralteter Zertifikate durchgeführt werden. Ein weiteres Ärgernis sind selbstsignierte Zertifikate, also solche Zertifikate $Cert_A = (ID_A, k_{pub.A}, s)$, deren Signatur sich Alice selber ausgestellt hat, ergo $s = Sig_{kprA}\,(h(ID_A, k_{pub}))$. Solche Zertifikate bieten keinerlei Vertrauenszugewinn und sind höchstproblematisch, da viele Nutzer eventuelle Warnmeldungen, dass ein selbstsigniertes Zertifikat eingesetzt wurde, bedenkenlos wegklicken.

Innerhalb des Teil 4 dieses Buches werden wir auf Zertifikate vorrangig in Zusammenhang mit dem mobilen Betriebssystem Android (siehe Abschn. 7.3.2) sowie den verschiedenen Ansätzen eines Zertifikats-Pinning (siehe Kap. 9) zu sprechen kommen.

2.4.8 Bloom-Filter

Ein Bloom-Filter [1] ist eine Datenstruktur, die bereits im Jahre 1970 von dem Mathematiker Burton Howard Bloom vorgeschlagen wurde und zur kompakten Darstellung einer endlichen Menge $A = \{a_1, a_2, \ldots, a_n\}$ von Elementen dient. Sie ist nicht im strengeren Sinne als ein kryptografischer Baustein zu verstehen, allerdings werden zur Konstruktion von Bloom-Filtern Hashfunktionen eingesetzt. Allgemein ist ein Bloom-Filter BF_A zu einer Menge A eine Bitfolge der Länge m Bits, wobei initial alle ihre m Bits auf 0 gesetzt sind. Sie dient oftmals als speicheroptimierter Repräsentant der eigentlichen Menge. Zur Erstellung eines Bloom-Filters für eine endliche Menge von Elementen werden k Hashfunktionen eingesetzt. Um ein Element a_i $i = 1, \ldots, n$ in das Bloom-Filter einzutragen, werden alle k Hashfunktionen auf das Element a_i angewandt, also $x_i = h_j(a_i)$ für $j = 1, \ldots, k$. Nun wird jedes Bit des Bloom-Filters BF_A von 0 auf 1 geändert, wenn es an dieser Position zu einem der so berechneten Hashwerte x_i korrespondiert. Dieser Vorgang wiederholt sich für alle n Elemente der Menge A. Ist dies geschehen, so ist BF_A als kompakte Darstellung der Menge A angelegt. Soll nun überprüft werden, ob ein Element $\tilde{a}$ tatsächlich zu der Menge A gehört, so müssen hierfür wiederum seine Hashwerte $\tilde{x} = h_j(\tilde{a})$ für $j = 1, \ldots, k$ berechnet werden und es ist zu prüfen, ob in BF_A alle korrespondierenden Bits auf 1 gesetzt sind. Sollte nur eines dieser Bits aus BF_A auf 0 gesetzt sein, so gilt mit Sicherheit $\tilde{a} \notin A$. Auf der anderen Seite ist festzuhalten, dass auch wenn tatsächlich alle zum Element $\tilde{a}$ korespondierenden Bits auf 1 gesetzt sind, es dennoch mit einer sehr geringen Wahrscheinlichkeit vorkommen kann, dass $\tilde{a}$ doch

nicht zu der Menge A gehört. Im Rahmen dieses Buches kommen wir in Kap. 12 auf eine Bloom-Filter-Variante im Zusammenhang mit Log-Dateien mobiler Nutzer zu sprechen. Allerdings erfolgt hierzu dann eine Anpassung von $x_i = h_j(a_i)$ nach $x_i = HMAC_{Kj}(a_i)$ für $j = 1, \ldots, k$. Anstelle von k unterschiedlichen Hashfunktionen wird also der Message Authentication Code HMAC verwendet, dies allerdings k-mal unter Nutzung von k unterschiedlichen Schlüsseln K_j. Warum diese kleine Änderung von Vorteil ist, werden wir dann zu gegebener Zeit ein wenig genauer erörtern.

2.4.9 Zusammenfassung

Lassen Sie uns zum Ende dieses Kapitels die geschilderten Grundlagen, die an einigen Stellen für das weitere Verständnis der in diesem Buch geschilderten Sicherheitslösungen sowie möglicher Angriffe notwendig sind, noch einmal zusammentragen.

Wir haben gängige Notationen zu den Kommunikationsparteien und Angreiferkategorien eingeführt und für die Belange der in diesem Buch noch zu schildernden Angriffe präzisiert. Wir haben ein Verständnis erlangt, was man auf dem Gebiet der Kommunikations- und Rechnernetze unter einem Protokoll versteht, dass es oftmals mit einem Handshake beginnt und dass das gleichzeitige Zusammenwirken mehrerer derartiger Protokolle in Kommunikationsschichten strukturiert aufgebaut ist. Des Weiteren ist uns ein wohlbekanntes Dilemma von Kommunikationsprotokollen, das Zwei-Armeen-Problem, geläufig und wir wissen, dass dieses Dilemma in der Praxis oftmals durch die Implementierung von Timern versucht wird zu lösen. Wir kennen hilfreiche unixoide Shell-Kommandos zur Prüfung der Erreichbarkeit eines Gerätes in IP-Netzen.

Aus dem Gebiet der Zahlentheorie haben wir uns mit dem chinesischen Restsatz vertraut gemacht. Wir wissen, dass mit dem Gauß-Algorithmus eine Möglichkeit existiert, effizient eine Lösung für das im chinesischen Restsatz aufgespannte Kongruenzsystem zu berechnen.

Darüber hinaus kennen Sie den prinzipiellen Aufbau sowie die Arbeitsweise und die gebotenen Schutzziele folgender elementarer kryptografischer Bausteine und Protokolle: Mit bitweisem XOR kann eine Nachricht verschlüsselt und auch wieder entschlüsselt werden, wobei die strikte Einhaltung einer Reihe von Bedingungen ausschlaggebend für das hierdurch tatsächlich erreichte Sicherheitsniveau ist. Sie können die Wirkungsweise von Hashfunktionen und Message Authentication Codes einordnen und unterscheiden. Sie wissen, dass Hashfunktionen neben der asymmetrischen Kryptografie ein wesentlicher Bestandteil bei der Bildung und Prüfung digitaler Signaturen sind. Aus der Klasse der MACs ist der HMAC anderen naiven Konstruktionen eines MAC vorzuziehen. Des Weiteren haben wir RSA als ein beispielhaftes asymmetrisches Verfahren kennengelernt, das man neben dem Verschlüsseln von Nachrichten, auch zum digitalen Signieren einer Nachricht verwenden kann. Bei den symmetrischen Verschlüsselungsverfahren können wir unterscheiden zwischen den Blockchiffren und den Stromchiffren. Wir haben verstanden, dass Zertifikate eine zwingend erforderliche Teilkomponente sind

damit asymmetrische Kryptografie überhaupt breit einsetzbar ist. Denn Zertifikate bieten ein technisches Mittel, den korrespondierenden öffentlichen Schlüssel zu einem privaten Schlüssel an seinen Eigentümer zu binden. In seiner technischen Umsetzung ist ein Zertifikat dann jedoch wiederum nichts anderes als eine digitale Signatur. Darüber hinaus haben wir uns vertraut gemacht mit der Schlüsselübereinkunft nach Whitfield Diffie und Martin Hellman. Diese dient der Aushandlung eines symmetrischen Schlüssels über einen unsicheren Kanal. Hierbei können wir einordnen, gegen welche Kategorie von Angreifern diese Schlüsselübereinkunft immun ist und bei welchen Angreifern das Protokoll versagt.

Nun haben wir das Rüstzeug zum Verständnis der nachfolgenden Kapitel erworben. Sicherlich können Sie auch dann getrost mit dem Lesen der nun folgenden Kapitel fortfahren, sollten Sie an dieser Stelle feststellen, dass Ihnen einzelne der hier noch einmal zusammengefassten Komponenten noch nicht ganz geläufig sind. Dann schlagen Sie bei Bedarf einfach noch einmal im Teil 2 ‚Grundlagen' nach.

Literatur

1. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
2. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory 22, 644–654 (1976)
3. Menezes, A.J., van Oorshot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
4. Paar, C., Pelzl, J.: Understanding Cryptography, A Textbook for Students and Practitioners. Springer, Heidelberg (2010)
5. Schneider, M.: Über Methoden der Generierung binärer Pseudozufallsfolgen zur Stromverschlüsselung. Dissertation, Shaker Verlag (1999
6. Smart, N.P.: Cryptography Made Simple – Information Security and Cryptography, S. 67–77. Springer, Cham (2016)
7. Tanenbaum, A.S.: Computer Networks, 5. Aufl. Prentice Hall, New Jersey (2010)
8. Trappe, W., Washington, L.: Introduction to Cryptography and Coding Theory, 2. Aufl. Pearson Prentice Hall, New Jersey (2006)

Teil II

Verwundbarkeiten drahtloser Kommunikationssysteme

Drahtlose Kommunikationssysteme werden für gewöhnlich eingeteilt in die Kategorien Mobiltelefonie, drahtlose lokale Netze und Personal Area Networks [1]. Im dritten Teil dieses Buches beschäftigen wir uns nun mit ausgesuchten Verwundbarkeiten und Angriffen zu jeder dieser drei Kategorien. Im Bereich der Mobiltelefonie beschäftigen wir uns mit der Verwundbarkeit mittels Malware auf GSM, UMTS sowie LTE und zeigen anschließend ausgewählte Aspekte einer 5G-Sicherheitsarchitektur auf. Die Verwundbarkeit schnurloser DECT-Telefonie wird ebenfalls erörtert. Innerhalb der zweiten Kategorie, der drahtlosen lokalen Netze, widmen wir uns Verwundbarkeiten verschiedener Wireless-LAN-Sicherheitsprotokolle der letzten 20 Jahre. Das führt uns bis zu WPA3 und zu IEEE-802.15.4 als einem weiteren Standard zur drahtlosen Kommunikation mit dem Augenmerk auf einer energiesparenden Kommunikation vorzugsweise zwischen eingebetteten Geräten. In der Kategorie der *Personal Area Networks* (PANs) als der dritten Säule beschreiben wir einige unter dem Namen Blueborne zusammengefasste Angriffe auf den Bluetooth-Standard sowie einen weiteren 2018 bekannt gewordenen Angriff zur Berechnung der Sitzungsschlüssel bei Verwendung von LE Secure Connection. Darüber hinaus interessiert uns die bei Bluetooth Low Energy (LE) vorzufindende Möglichkeit, das Erfassen von Bewegungsprofilen zu unterbinden, sowie deren Belastbarkeit. Wir beschließen dieses Kapitel, nicht ohne auf die zunehmend an Bedeutung gewinnende *Near Field Communication* (NFC), deren Sicherheitsstandards sowie mögliche Verwundbarkeiten einzugehen.

Die Abb. II.1 bietet eine Klassifikation mobiler drahtloser Netze, wobei diejenigen Technologien, für die wir beispielhafte Verwundbarkeiten und Angriffe beschreiben, hervorgehoben sind.

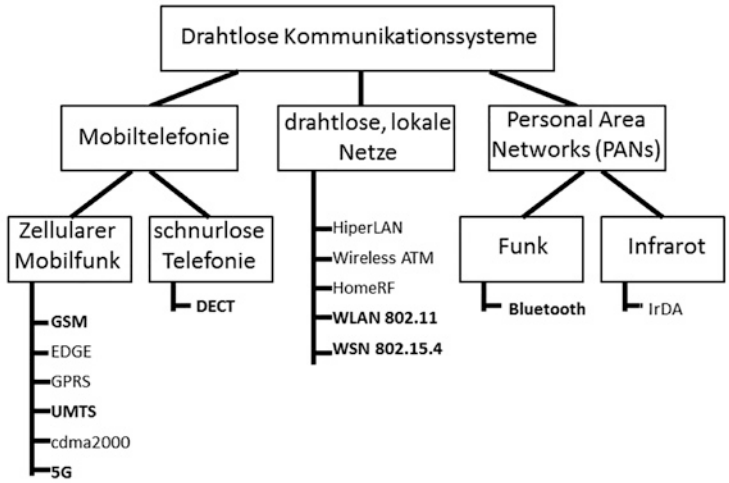

Abb. II.1 Klassifikation drahtloser Kommunikationssysteme. (Quelle: eigene Darstellung in Anlehnung an [1])

Literatur

1. Eckert, C.: IT-Sicherheit: Konzepte – Verfahren – Protokolle. De Gruyter Studium (2006)

Verwundbarkeiten in drahtlosen lokalen Netzen

3

3.1 Grundsätzliche Bemerkungen zur Funktionsweise der Sicherungsschicht

Wir beginnen damit, welche Möglichkeiten der Manipulierbarkeit von Daten es auf den untersten Übertragungsschichten des ISO/OSI-Stacks gibt. Hierzu ist es jedoch erforderlich, sich erst einmal den prinzipiellen Abläufen zu widmen *ohne* dass ein Angreifer das System korrumpieren wollte. Der einzige widrige externe Einfluss ist damit die Beschaffenheit des gemeinsamen Übertragungsmediums, welches im Falle von Luft doch relativ störbehaftet ist. Diese Tatsache, es mit einem rauschbehafteten Medium zu tun zu haben, wird für gewöhnlich beim Entwurf und bei der Entwicklung der hierüber zum Einsatz kommenden Übertragungsprotokolle stark berücksichtigt.

Um ein genaueres Verständnis ausgewählter Angriffe auf Protokolle der Sicherungsschicht zu erlangen, kann es somit nicht schaden, einige grundlegende Aspekte der Sicherungsschicht aufzufrischen. Die Sicherungsschicht widmet sich der gesicherten Übertragung von Informationen zwischen benachbarten Systemen, also solchen die keine weiteren zwischengelagerten Netzkoppelelemente zur Übertragung der Daten benötigen. Neben der Strukturierung von Bits in Wörter und Rahmen enthalten die Protokolle der Sicherungsschicht Komponenten zum Multiplexing, der Fehlererkennung, der Flusskontrolle, sowie der Fehlerbehandlung und Korrektur. Diejenigen Leser, die mit der prinzipiellen Arbeitsweise der Sicherungsschicht schon vertraut sind, können also direkt mit dem Abschn. 3.2 (Verwundbarkeit von IEEE 802.15.4) fortfahren.

Welchen genauen Dienst die Sicherungsschicht sinnvollerweise der nächsthöheren Schicht bereitstellt, hängt stark von dem zur Verfügung stehenden Übertragungsmedium ab. Über die Luftschnittstelle wird typischerweise ein *bestätigter verbindungsloser Dienst* angeboten, es erfolgt also kein Verbindungsaufbau. Es wird allerdings hier

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_3

jeder empfangene Rahmen – dies ist der Begriff, der sich für die Dateneinheiten auf der Sicherungsschicht etabliert hat – bestätigt.

Im Einzelnen behandelt die Sicherungsschicht folgende Punkte:

Die Rahmenbildung, das *Framing,* unterteilt den rohen Bitstrom in Rahmen. Hierbei kommt erschwerend hinzu, dass ein roher Bitstrom häufig nicht frei von Übertragungsfehlern (Bitdrehern) auf dem störbehafteten Medium ist. Die Unterteilung des Datenstroms in einzelne Bitfolgen wird auf der Senderseite durch ein *Flag-Byte* erzeugt. Dieses könnte zum Beispiel die Bitfolge 01111110 sein. Nach diesen Flag-Bytes sucht empfängerseitig der hierfür zuständige Prozess der Sicherungsschicht um aus den eingehenden Rahmen wiederum den durchgehenden Bitstrom zu extrahieren. Nun kann es jedoch vorkommen, dass die Nutzdaten, die in einzelne Rahmen zu untergliedern sind, exakt eine Bitfolge enthalten, die der des vereinbarten Flag-Bytes entspricht. Damit der empfängerseitige Prozess der Sicherungsschicht nicht fälschlicherweise an dieser Stelle eine Rahmentrennung durchführt, werden vor dem Versenden derartige Passagen in dem Nutzdatenstrom mit dem sogenannten *Bit-Stuffing* aufgefüllt. So können die Grenzen zwischen zwei Rahmen empfängerseitig eindeutig erkannt werden und fälschliche Trennungen aufgrund einer identischen Bitfolge wie der des Flag-Bytes in der Nutzdaten-Bitfolge elegant unterbunden werden.

Des Weiteren sind auf der Sicherungsschicht Übertragungsfehler auf dem drahtlosen Medium zu erkennen und wenn möglich zu beheben. Dabei tendieren Bitübertragungsfehler auffällig oft dazu, in Bündeln aufzutreten. Verfahren zur *Fehlererkennung* spendieren gerade einmal so viel Redundanz für die Datenübertragung, dass ein Übertragungsfehler aus einer bestimmten Fehlerklasse erkannt wird und die wiederholte Datenübertragung im Fehlerfalle anlaufen kann. Ist das Protokoll der Sicherungsschicht nun so ausgelegt, dass es eine wiederholte Datenübertragung umgehen soll, so ist deutlich mehr Redundanz innerhalb der zu übertragenden Daten einzubauen, damit empfängerseitig eigenständig eine *Fehlerbehebung* möglich wird. Damit kann das Ziel verfolgt werden, empfängerseitig eigenständig auf den korrekten Inhalt eines Datenblocks zu schließen, auch wenn dieser fehlerhaft beim Empfänger angekommen sein sollte.

Um nun den in Abschn. 3.2 beschriebenen Angriff nachvollziehbar zu machen, ist es vorteilhaft, zu verstehen, dass nicht jede mögliche Bitfolge ein korrektes *Codewort* aus einer vorab festgelegten Menge von Codewörtern ist. Hierbei spielt der Begriff der *Hamming-Distanz* eine wichtige Rolle. Hiermit ist die Anzahl der unterschiedlichen Bits zwischen zwei Codewörtern gemeint. So würden beispielsweise die beiden erlaubten acht Bit langen Codewörter 00001111 und 00111100 die Hamming-Distanz vier aufweisen, da sich ihre Bits gegenseitig an der dritten, vierten, siebten und achten Stelle unterscheiden. Nun ist es allerdings so, dass typische Protokolle der Sicherungsschicht in den einzelnen Rahmen nur Bitfolgen erlauben, die aus einer vorab festgelegten Menge von erlaubten Codewörtern stammen. Dies verdeutlichen wir an einem einfachen Beispiel: Erlaubt seien nur Codewörter der Länge zehn und davon auch nur solche mit einer Hamming-Distanz von fünf. Zwei gültige Codewörter wären also das Codewort 0000011111

und das Codewort 0000000000. Ein Protokoll der Sicherungsschicht mit Fehlerbehebung würde nun erkennen, dass ein empfangenes Codewort dieser Bauart wie beispielsweise das Codewort 0000000111 nicht zu der Menge der gültigen Codewörter gehört. Ein empfängerseitiger Prozess, der derartige Verfahren der Sicherungsschicht abbildet, würde nun von einem Übertragungsfehler auf dem rauschbehafteten Medium ausgehen und die *Fehlerbehebung* durchführen. Er würde das ungültige Codewort 0000000111 in dasjenige gültige Codewort überführen, das zum übertragenen Codewort die kleinste Hamming-Distanz aufweist. Da das gültige Codewort 0000011111 zur empfangenen Bitfolge 0000000111 die Distanz zwei aufweist, und das gültige Codewort 0000000000 zu 0000000111 seinerseits eine Distanz von drei hat, würde die empfängerseitige Fehlerbehebung in diesem Falle die Bitfolge in das gültige Codewort 0000011111 überführen.

Durch diese Vorüberlegungen wird auch klar, warum bei einer Bitfolge der Länge n längst nicht alle 2^n möglichen Wörter als erlaubte Codewörter verwendet werden können. Allgemein lässt sich Folgendes sagen: Will man den Fehlerbehebungsmechanismus der Sicherungsschicht in die Lage versetzen, d Bitdreher aufgrund von Übertragungsfehlern zu beheben, muss die Distanz zwischen allen gültigen Codewörtern mindestens $2d+1$ betragen. Denn dann ist auch bei bis zu d Bitübertragungsfehlern pro Codewort das ursprünglich versendete gültige Codewort noch näher als jedes andere gültige Codewort. Allerdings bedeutet dies für unser obiges Beispiel auch, dass bei einer anderen empfangenen Bitfolge wie beispielsweise von 0000000011 diese in das gültige Codewort 0000000000 überführt würde. Solange die Anzahl der Bitfehler pro Codewort also unter der durch die Hamming-Distanzen aller gültigen Codewörter festgelegten Schranke liegt, ist sichergestellt, dass die Fehlerbehandlung die eingehende Bitfolge immer in dasjenige gültige Codewort überführt, welches auch tatsächlich senderseitig übertragen worden ist. Liegt die Anzahl der Bitdreher aufgrund einer fehlerhaften Übertragung über das rauschbehaftete Medium jedoch darüber, so ist diese Garantie nicht mehr gegeben und die eingehende Bitfolge wird fälschlicherweise in ein ursprünglich gar nicht gesendetes Codewort überführt.

Mit diesen Vorüberlegungen zu allgemeinen Entwurfsfragen der Sicherungsschicht, insbesondere zur Bereitstellung eines *bestätigten verbindungslosen Dienstes* für die nächst höhere Protokollschicht, sollten wir in der Lage sein den nachfolgend geschilderten Angriff besser zu verstehen.

3.2 Verwundbarkeit von IEEE 802.15.4

Den nun geschilderten Paket-in-Paket-(PiP-)Injizier-Angriff wollen wir beispielhaft anhand des Übertragungsstandards IEEE 802.15.4 erörtern. Wir haben diesen Angriff nicht deshalb gewählt, weil das Schadenspotenzial so beachtlich groß ist. Tatsächlich gibt es in vielen bestehenden Systemen deutlich einfachere Wege eines Angriffs. Vielmehr ist dieser Angriff für den ‚Feinschmecker' technischer Angriffe gedacht, da er eben nicht in *Brute-Force* Manier ans Ziel führt, sondern die Angreifer zur Umsetzung

eines derartigen Angriffs über einiges an Protokollwissen verfügen müssen und das Zusammenspiel der einzelnen Bausteine der Sicherungsschicht auf einem beachtlichen Detailierungslevel verinnerlicht haben. Wie so oft gilt dies für die Entdecker der Schwachstelle. Diejenigen, die sie dann anwenden, benötigen dieses Detailwissen nicht, um an ihr Ziel zu kommen, da sie nur noch in der Lage sein müssen, die entsprechenden Werkzeuge zu bedienen. Dem Angriff liegt das in Abb. 3.1 dargestellte Szenario zugrunde, bei dem Mallory eine Nachricht a mittels des drahtlosen Übertragungsstandards IEEE 802.15.4 an das Opfer sendet.

IEEE 802.15.4 ist das drahtlose Übertragungsprotokoll für WPANs (Wireless Personal Area Networks) und wird oftmals auf den höheren Protokollschichten zusammen mit ZigBee [19] verwendet. Es wird zur drahtlosen Kommunikation zwischen ‚Dingen' im Internet der Dinge, für drahtlose Personal Area Networks sowie zur Kommunikation in drahtlosen Sensornetzen verwendet. Für das Einsatzgebiet Smart Home wurde 2012 auf der DeepSec vorgeführt, dass die Schlüsselverteilung bei ZigBee als hochgradig unsicher zu bewerten ist. Doch nun wenden wir uns unserem PiP-Angriff auf IEEE 802.15.4 zu.

Travis Goodspeed [5] und weitere Co-Autoren [6, 7] haben diesen Angriff erstmalig 2011 auf dem USENIX Security Symposium in San Francisco vorgestellt. Auf der Konferenz Troopers 2015 in Heidelberg hatte ich dann die Gelegenheit, persönlich Kontakt mit dieser Gruppe aufzunehmen. Hieraus ist dann als Proof of Concept (PoC) die praktische Projektarbeit [3] unter Verwendung von GoodFET ApiMotes und KillerBee entstanden.

Prinzipiell sind PiP-Angriffe auch für weitere Protokolle der Sicherungsschicht und auch für höhere Protokollschichten denkbar. Hierbei ist bemerkenswert, dass wie im konkreten Fall von IEEE 802.15.4 die notwendigen Manipulationen durch einen Angreifer keinerlei privilegierte Rechte erfordern. Konkret reicht es Mallory aus, auf dem senderseitigen System Nutzerrechte zu haben. Das Erlangen von weiteren Systemrechten wie Kernelrechten ist somit nicht zwingend erforderlich.

Aus Abschn. 3.1 wissen wir bereits, dass die Einteilung einer Bitfolge in Rahmen eine Implementierung von Algorithmen erfordert, die das Framing umsetzen. Die bewährte Darstellungsform einzelner Felder aus den IEEE- und auch IETF-Spezifikationsdokumenten für Adressen, Nutzdaten, Prüfsummen usw. in Form von ‚Kästchen', deren Größe die Anzahl der hierfür reservierten Bits repräsentiert, ist also nicht wirklich so strikt, wie uns diese Darstellungen glauben machen könnten, und daher mit einer gewissen Vorsicht zu genießen. Gleiches gilt natürlich auch für die Darstellung der Bitübertragungsschicht (engl. Physical Layer), kurz PHY, sowie innerhalb der Sicherungsschicht der Medium-Access-Control-Schicht, kurz MAC-Schicht, im

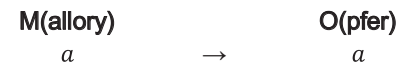

Abb. 3.1 PiP mit Mallory und Opfer. (Quelle: eigene Darstellung)

IEEE 802.15.4 Standard. Travis Goodspeed hat hierfür den Begriff *phantom boundaries* geprägt [5]. Und schlimmer: Diese durch die ‚Kästchen'-Darstellung suggerierte strikte Abgrenzung von Köpfen mit Quell- und Zieladressen, Nutzdaten und Prüfsummen in den Standarddokumenten kann als ein stillschweigend hingenommener Vertrauensvorschub hinsichtlich der Integrität und des Ursprungs der Daten interpretiert werden. Natürlich ist dies technisch durch nichts gerechtfertigt.

Die Abb. 3.2 zeigt eine schematische Darstellung der Sicherungsschicht und der MAC-Schicht für IEEE 802.15.4 in der besagten ‚Kästchen'-Darstellung unter Angabe der für jeden Bereich vorgesehenen Anzahl an Bytes. In abgekürzter Form sind hierbei die Bereiche für *Frame Control Field* (FCF), *Data Sequence Number* (DSN), *Frame Check Sequence* (FCS) und *Start Frame Delimiter* (SFD) aufgeführt.

Doch nun zur wesentlichen Idee eines PiP-Angriffs: Unter gewissen Umständen kann ein Bitdreher in einem ‚äußeren' Rahmen dazu führen, dass ein in diesem Rahmen enthaltener bzw. vorsätzlich eingebauter ‚innerer' Rahmen empfängerseitig als der ‚äußere' Rahmen interpretiert wird. Ist dies der Fall, so reicht die Sicherungsschicht diesen inneren Rahmen an die Protokolle der höheren Schicht weiter.

Wofür soll das gut sein? Nun ja, man kann sich leicht vorstellen, dass sich hierdurch neben anderen Dingen die Möglichkeit ergibt, Rahmen an einer Firewall oder aber an einem *Network-Intrusion-Detection-System* ((N)IDS) bzw. *Network-Intrusion-Prevention-System*((N)IPS) vorbei zu schleusen.

Für gewöhnlich erwarten Protokolle der Sicherungsschicht Bitfolgen, die unterschiedlich zu interpretieren sind. Der Bereich der *Präambel* innerhalb einer zu übertragenden Bitfolge dient zum Aufwecken der Radio-Komponente. Der nachfolgende *Sync*-Bereich ist aus zweierlei Gründen erforderlich: Zum einen dient er der Zeitsynchronisation zwischen Sender und Empfänger. Zum anderen ist er erforderlich, um eine Unterscheidung zwischen tatsächlich gültigen eingehenden Daten und einem Hintergrundrauschen auf der Übertragungsstrecke treffen zu können. Der *Rumpf* enthält die eigentlichen Nutzdaten einschließlich all derjenigen Informationen über Protokolle der nächst höheren Protokollschicht.

Eine entscheidende Voraussetzung für das Gelingen des Angriffs ist die Beobachtung, dass alle drei Regionen, also Präambel, Sync-Bereich und Rumpf, die gleiche Symbolmenge

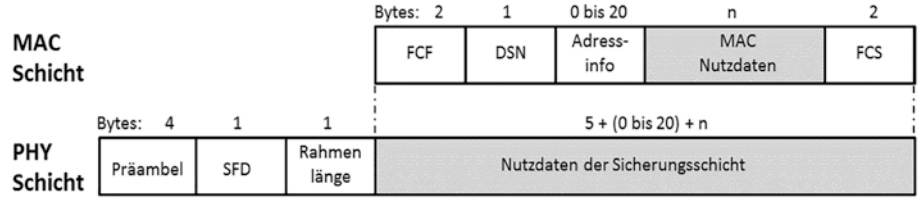

Abb. 3.2 Schematische Darstellung der Datenformate der PHY- und der MAC-Schicht für IEEE 802.1.5.4. (Quelle: eigene Darstellung in Anlehnung an [5])

verwenden. Dies bedeutet, dass eine Präambel und der Sync-Bereich prinzipiell auch innerhalb eines Rumpfes verbaut werden können. Entsprechend des IEEE-802.15.4-Standards könnte ein gültiger Rahmen in Hex-Notation das in Abb. 3.3 gezeigte Aussehen haben:

Allerdings wäre es auch denkbar, innerhalb des Rumpfes die für die Präambel (00 00 00 00) und den Sync-Bereich (a7) vorgesehenen Symbole zu verwenden, wie in Abb. 3.4 zu sehen.

Nun haben wir bereits erörtert, dass Symbolfehler bei der Übertragung überraschend häufig auftreten. Dies kann ein Angreifer versuchen auszunutzen, indem er viele Rahmen der in Abb. 3.4 dargestellten über Broadcast verbreitet. Er verbindet damit die Hoffnung, dass ein Symbolfehler die äußere Präambel oder den äußeren Sync-Bereich überschreibt und somit empfängerseitig nur die innere Präambel und Sync als Start interpretiert werden, sodass allein die Bitfolge danach als gültiger Rumpf des IEEE-802.15.4-Rahmens ausgewertet wird. Doch ganz so einfach macht der Standard einem Angreifer diesen Angriff doch nicht. Denn laut Spezifikation ist die Hex-Folge a7 innerhalb des Rumpfes nicht erlaubt. Der Angreifer muss also ein wenig cleverer vorgehen.

Tatsächlich enthalten die Rahmen aus den Abb. 3.3 und 3.4 noch keine in sich konsistente Information, sodass sie empfängerseitig verworfen werden würden. Denn der Bereich des Rumpfes enthält immer auch Informationen zur Länge des Rahmens, Kopf, PAN, MAC und der *Prüfsumme*. Des Weiteren wird jedes mögliche Hex-Symbol, also 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, in Form einer 32-Bitlangen Bitfolge, eines sogenannten Chips, kodiert. Beispielsweise ist das Hex-Symbol 0 die Bitfolge aus der Abb. 3.5,

sodass die Präambel sich aus acht Chips dieser Art zusammensetzt. Ähnliches gilt für die beiden Hex-Werte a und 7, die ebenfalls jeweils als 32-Bit-Chips kodiert sind (Abb. 3.6):

Präambel	Sync	Rumpf
00 00 00 00	a7	0f 01 08 82 ff ff ff ff

Abb. 3.3 Beispielhafter IEEE-802.15.4-Rahmen. (Quelle: eigene Darstellung in Anlehnung an [5])

Präambel	Sync	Rumpf
00 00 00 00	a7	0f 00 00 00 a7 ff ff ff

Abb. 3.4 Beispielhafter IEEE-802.15.4-Rahmen mit eingefügten inneren Rahmen. (Quelle: eigene Darstellung in Anlehnung an [5])

Hex	Chip
0	11011001110000110101001000101110

Abb. 3.5 32-Bit-Chipfolge des Hex-Wertes 0 laut IEEE 802.15.4. (Quelle: eigene Darstellung)

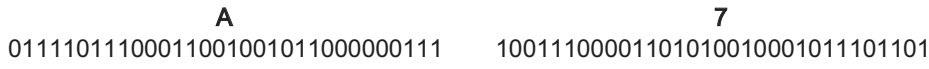

Abb. 3.6 2×32-Bit-Chipfolgen des a7-Sync-Bereiches eines IEEE-802.15.4-Rahmens. (Quelle: eigene Darstellung)

Und nun kommt endlich der eigentliche Trick des Paket-in-Paket-Injizier Angriffs auf den 802.15.4-Standard: Was wäre, wenn man den empfängerseitigen Mechanismus zur Fehlerbehebung ausnutzen könnte, um dennoch a7-Folgen in den Rumpf eines IEEE-802.15.4-Rahmens einzuschleusen? In Abschn. 3.1 haben wir ja bereits beschrieben, wie ein empfangenes ungültiges Codewort durch die Fehlerbehebung in dasjenige gültige Codewort überführt wird, das die kürzeste Hamming-Distanz zu dem empfangenen Codewort hat. Und tatsächlich haben Travis Goodspeed und Co-Autoren [5] einen Angriff durch Ausnutzung des Fehlerbehebungsmechanismus durchführen können, indem sie anstelle von 00000000a7 die Hex-Folge 11111111b0 in den Rumpf des Rahmens eingefügt haben. Da jeweils die Hamming-Distanz des ungültigen *Codeworts* 1 am nächsten zum gültigen Codewort 0, die des ungültigen Codeworts b am nächsten zum gültigen Codewort a und die des ungültigen Codewortes 0 am nächsten zum gültigen Codewort 7 ist, wird die empfängerseitige Fehlerbehandlung ausgenutzt, um einen inneren Rahmen beginnend mit 00000000a7 an der eigentlichen Filterung durch die Sicherungsschicht vorbei zu schleusen. Oder noch ein wenig konkreter: Ein weiterer Präambel-Bereich und Sync-Bereich im Rumpf eines übertragenen IEEE-802.15.4-Rahmens ist bei der Übertragung des Rahmens noch gar nicht explizit vorhanden. Er wird tatsächlich erst empfängerseitig aufgrund des dort aktivierten Algorithmus zur Fehlerbehebung erzeugt und kann so durch die vorgeschaltete Filterung auch nicht erkannt werden.

Dieser Sachverhalt ist in der Abb. 3.7 noch einmal dargestellt: Auch wenn am Opfergerät der von Mallory präparierte Rahmen fehlerfrei ankommt, so wandelt die Komponente zur Fehlerbehebung die Folge 11111111b0 in die Folge 0000000a7 um. Dies hat zur Konsequenz, dass fälschlicherweise allein **Y** an die nächst-höhere Protokollschicht gereicht wird und nicht der gesamte übertragene Rumpf **X**11111111b0**Y** des Rahmens. Bitte verstehen Sie die Parameter **X** und **Y** hierbei als Platzhalter für weitere im Rumpf des Rahmens enthaltene Hex-Folgen.

Anhand der Beschreibung dieses Injizier-Angriffs lässt sich meiner Meinung nach sehr schön erkennen, auf welchem Detaillierungslevel die Vorbereitung, die Durchführung und die eventuell möglichen Gegenmaßnahmen von heutigen Angriffen auf drahtlose Kommunikationsprotokolle zu durchdringen sind. Man muss sich klarmachen: Wäre das Sicherungsprotokoll so umgesetzt worden, dass die Filterung nach ausgewiesenen Zeichen im Rumpf des Rahmens wie a7 *nach* der Fehlerbehebung erfolgen würde, so wäre dieser Angriff nicht möglich. Allerdings muss man sich auch vergegenwärtigen, dass bei der Entwicklung der Sicherungsschicht erst einmal keinerlei Expertise zu IT-Sicherheit gefragt ist, sondern vor allem Expertise aus dem Bereich

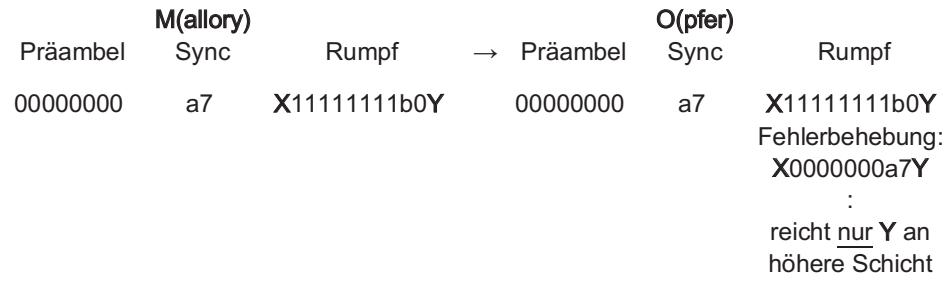

Abb. 3.7 PiP-Angriff mit Mallory und Opfer unter Ausnutzung der Fehlerbehebung. (Quelle: eigene Darstellung)

Codierungstheorie sowie ein generelles Verständnis der Übertragungstechniken auf den untere PHY- und MAC-Schichten. Dies dürfte sich aber mit dem Wissen um die Verwundbarkeit von Protokollen der Sicherungsschicht durch PiP-Injizier-Angriffe nachhaltig geändert haben.

▶ Der PiP-Injizier-Angriff auf ein Opfergerät welches zur drahtlosen Kommunikation den Standard IEEE 802.15.4 verwendet, macht sich zunutze, dass auf der Sicherungsschicht die Komponenten zur Fehlerbehandlung und zur Filterung unerwünschter Muster im Rumpf des Rahmens in der falschen Reihenfolge ausgeführt werden. Die Verwundbarkeit resultiert damit aus einer **unklaren Spezifikation der PHY- und MAC-Schicht, die bei der technischen Umsetzung zu frei ‚ausgestaltet' werden kann.**

3.3 Maßnahmen gegen einen Paket-in-Paket-Injizier-Angriff

3.3.1 Byte-Stuffing

Als Maßnahme gegen den geschilderten PiP-Injizier-Angriff haben Anshuman Biswas und Co-Autoren [2] einen Ansatz unter der Verwendung von *Byte-Stuffing* vorgeschlagen. Dieses Verfahren macht Gebrauch von dem in Abschn. 3.1 erörterten Konzept des Bit-Stuffing. Allerdings propagieren die Autoren Byte-Stuffing, das folgendermaßen funktioniert:

Auf der Senderseite durchsucht (scannt) der Sender die Nutzdaten nach ‚00'-Symbolen und füllt bei Bedarf mit ‚ff'-Symbolen auf. Dieser Vorgang erfolgt unabhängig davon, ob die Sequenz gefolgt ist von ‚a7' oder nicht.

Empfängerseitig werden die Nutzdaten nun ebenfalls durchsucht, diesmal nach ‚00'-Symbolen gefolgt von ‚ff'-Symbolen. Die empfängerseitige Middleware entfernt nun das Symbol-‚ff', bevor es an die höhere Schicht gereicht wird.

Da Byte-Stuffing nicht Teil des 802.15.4-Standards ist, haben Biswas und Co-Autoren dessen Funktionalität in der Middleware umgesetzt und dort evaluiert: In einem Multi-Hop-Netzwerk hat es so gut wie keinen negativen Einfluss auf die Nettodatenrate. Allerdings beeinflusst es die Latenz, sodass pro Übertragung schätzungsweise fünf Millisekunden mehr senderseitige und empfängerseitige Bearbeitung anfallen.

Dass Byte-Stuffing nur bedingt erfolgreich zur Abwehr gegenüber PiP-Injizier-Angriffen ist, sollte offenkundig sein. So hat die Gruppe um Goodspeed auch recht schnell reagiert und Anti-Byte-Stuffing propagiert, indem der Angreifer die Folge f0 00 00 00 0a anstelle der Folge 00 00 00 00 a7 in die Nutzdaten einfügt.

3.3.2 Verwendung der in IEEE 802.15.4 vorgesehenen AES-Varianten

Zum Ende dieses Abschnitts über PiP-Injizierung folgen hier noch einige Anmerkungen zur Nutzung des IEEE-802.15.4-Standards unter Verwendung der dafür vorgesehenen kryptografischen Bausteine [13]. Hierfür sieht der IEEE-802.15.4-Standard die *Security Layer* vor. Diese ist ansprechbar über die Anwendungsebene, indem Sicherheitsparameter spezifiziert werden, wobei wie so oft auch hier die Voreinstellung ‚keine-Sicherheit' vorzufinden ist.

Des Weiteren hat der Anwender die Wahl zwischen *nur Verschlüsselung* (AES-CTR), *nur Authentifizierung* (AES-CBC-MAC), sowie *Verschlüsselung und Authentifizierung* (AES-CCM). Alle drei Varianten basieren also auf der 128-Bit-Blockchiffre Advanced Encryption Standard (AES). Im ersten Fall zur Verschlüsselung mit dem blockweisen Verkettungsmodus Counter Mode, im zweiten Fall zur Authentifizierung als Baustein für einen Message Authentication Code und im dritten Fall im *Counter-with-CBC-MAC* (CMC) Mode, der gleichzeitig eine Verschlüsselung und eine Authentifizierung bietet.

Was bedeuten diese Varianten nun für die Abwehr eines Paket-in-Paket-Injizier-Angriffs? Nun, der PiP-Injizier-Angriff funktioniert auf jeden Fall unter der Voreinstellung *‚keine Sicherheit'*. Er funktioniert des Weiteren aber auch unter Verwendung der Sicherheitsvariante *‚nur Authentifizierung'* (AES-CBC-MAC). In Kombination mit allen weiteren Varianten dürfte er nicht mehr erfolgreich durchführbar sein.

Allerdings ist darüber hinaus zu bedenken, dass es bei einer Verschlüsselung mit der Blockchiffre AES aufgrund des Auffüllens zu einem Vielfachen von 128-Bit-Einheiten zu einem größeren Datenvolumen auf der Funkstrecke kommt. Für energiesensible Anwendungen, für die befürchtet wird, hierdurch die Lebenszeit eines eingebetteten Gerätes empfindlich zu reduzieren, kann dies im Einzelfall durchaus dazu führen, bewusst auf eine Verschlüsselung der zu übertragenden Daten zu verzichten. Zumal beispielsweise der getastete Messwert eines Sensors typischerweise nicht mehr als zwei Byte zur Übertragung einer derartigen Information benötigt, sein verschlüsselter Wert jedoch 16 Byte inklusive des Paddings erfordern würde.

3.4 Anmerkungen zur Analyse proprietärer Protokolle

Manchmal ist es wünschenswert oder einfach erforderlich, proprietäre drahtlose Übertragungsprotokolle zu analysieren, also solche deren Funktionsweise und Aufbau man nicht genau kennt bzw. die nicht als standard-konforme Spezifikation vorliegen. Dies geschieht zum Beispiel, um den Aufbau eines Rahmens oder eines Paketes nachvollziehbar zu machen, oder, um selbst Rahmen mit eigenen Inhalten bzw. Befehlsfolgen in das System einzuschleusen, falls das Protokoll neben einer gezielten Verschleierung (Obfuskierung) aufgrund des proprietären Charakters des Übertragungsprotokolls nicht weiter gesichert wäre.

Bewährt hat sich hierbei das Mitschneiden von Verkehr von der in unserem Fall drahtlosen Ethernet-Schnittstelle. Dabei ist darauf zu achten, dass sich das System im *Promiscuous Mode* befindet, damit auch Rahmen mitgeschnitten werden, die nicht an den eigenen Rechner adressiert sind, deren sendendes Gerät sich jedoch im Empfangsbereich befindet.

Liegt der so mitgeschnittene Verkehr dann als .cap-Datei vor, so kann dieser in ein Netzwerkanalyse Werkzeug wie *Wireshark* eingelesen und weiter analysiert werden. Da Wirershark jedoch eine Filterung nur auf bekannten Protokollen ermöglicht, wie beispielsweise auf ICMP, TCP, UDP oder eben auch IEEE 802.15.4, so ist es notwendig sich seine eigene Erweiterung für das proprietäre Protokollformat zu implementieren und auf diese Weise einen *Dissektor* anzulegen. Hierzu verwendet man das Plug-in für Protokoll-Dissektoren und erstellt seinen eigenen Dissektor unter Verwendung der Scriptsprache *Lua.* Nutzt man des Weiteren Werkzeuge zur Analyse regulärer Ausdrücke wie beispielsweise *RegExr* oder *regex1o1,* so lässt sich hierüber oftmals relativ komfortabel auf den Präambel- und Sync-Bereich eines erfassten Rahmens schließen, ohne mit dessen konkretem Aufbau vollständig vertraut zu sein. Ist dies erfolgt, so lassen sich weitere einzelne Nachrichtentypen des zu analysierenden Protokolls ermitteln, bis schließlich das gesamte Protokollformat und im Idealfall das vollständige Protokollverhalten offengelegt ist. Den Startpunkt bildet jeweils ein leerer Dissektor der dann in das Wireshark-Plugin-Verzeichnis aufzunehmen und danach Schritt für Schritt an das proprietäre Protokoll anzupassen ist.

```
proprietaer_proto=Proto(''labor'',''Proprietaeres Protocol'')
   function proprietaer_proto.dissector(buffer,pinfo,tree)
        pinfo.cols.protocol=''LABOR''
        local subtree =
              tree:add(proprietaer_proto,buffer(),''LABORProtocol-
Data'')
        …
        …
        …
   end
   udp_table=DissectorTable.get(''udp.port'')
   udp_table:add(9999,labor_proto)
```

3.5 Verwundbarkeiten von WLAN

Von einem akademischen Blickwinkel aus betrachtet kann man fast schon dankbar sein, wenn man sich die Historie in der Evolution der Sicherheitsarchitekturen des WLAN-802.11-Standards rückblickend vor Augen führt. Denn hier ist vieles geboten, was als Lehrstück dienen kann. So lässt sich recht anschaulich dokumentieren, auf welchen Ebenen und in welchen Phasen eines Entwicklungsprozesses sich neue Verwundbarkeiten aufgrund einer lückenhaften Spezifikation, der Bereitstellung von abwärtskompatiblen Lösungen, fehlerhafter Implementierungen oder aber des Roll-Outs in eine weltweit genutzte Technologie einnisten können. Doch bevor wir auf die Ursprünge mit Wired Equivalent Privacy (WEP) und seine dann nachfolgenden Verbesserungen durch WiFi Protected Access (WPA) sowie IEEE 802.11i (-WPA2) und WPA3 eingehen, möchte ich noch einmal auf den elementaren Sicherheitsbausteinen von WEP hinweisen, den XOR. Diesen hatten wir bereits in Abschn. 2.4.1 eingeführt. Auch wenn wir wissen, dass WEP unter keinen Umständen mehr zu verwenden ist und seit 2004 mit IEEE 802.11i eine deutliche verbesserte Variante zur Verfügung steht, zumindest was deren kryptografische Bausteine anbelangt, so bieten sich hieraus doch reichlich Anhaltspunkte für ein Buch, das die Verwundbarkeit und Schwachstellenanalyse drahtloser, mobiler IT-Systeme aufarbeitet. So hat es sich gerade für die Architektur neuer Sicherheitslösungen bewährt, sich mit den Schwachstellen älterer Lösungen auseinanderzusetzen und deren Angriffsszenarien nachvollziehbar zu machen.

3.5.1 Designschwächen von WEP

‚Bitte ein Bit für den *IV*' ist kein zufällig aufgeschnappter Gesprächsfetzen in einer Kneipe im Eifler Raum, sondern plakatiert eine fiktive Diskussion innerhalb einer Standardisierungssitzung, bei der sich vermutlich die Fraktion der Sicherheitsarchitekten gegen andere Fraktionen mit anderen Interessen zu behaupten hatte. Für die Vertreter letzterer Gruppe schien die zu erbringende Performanz, also die Nettodatenrate auf dem Übertragungsmedium, ein höheres Gut zu sein als die gebotene Sicherheit des zu entwickelnden Sicherheitsprotokolls. Denn dem *IV,* also dem Initialisierungsvektor, obliegt eine hohe Verantwortung hinsichtlich der Sicherheit von WEP, wie wir sehen werden.

Solch eine kontroverse Diskussion innerhalb des Standardisierungsgremiums ist schon verwunderlich, führt man sich noch einmal vor Augen, dass es sich bei der Informationssicherheit nicht um einen netten Zusatz handelt, sondern dass das Protokoll *Wired Equivalent Privacy* (WEP) ja einzig zur Gewährleistung der Schutzziele Vertraulichkeit, Integrität und Authentizität der zu übertragenden Daten ins Leben gerufen wurde.

Doch der Reihe nach: Im Einzelnen verhalten sich Sender und Empfänger wie folgt:

Am Sender, wenn eine Klartext-Bitfolge *X* übertragen werden soll, geschieht Folgendes:

1. Berechnung einer Saat S aus der Zusammenführung von IV und K, also $S = IV||K$.
2. Die Saat S eingespeist in einen *Pseudorandom Number Generator (PRNG),* erzeugt einen Schlüsselstrom:

$$KS := k_0, k_1, \ldots, k_{n-1} \leftarrow PRNG(S)$$

3. Aus der Klartext-Bitfolge $X := x_0, x_1, \ldots, x_{n-1}$ wird ein *Integrity Check Value (ICV)* mittels eines *Cyclic Redundancy Check (CRC)* berechnet; beide werden anschließend zusammengesetzt: $X||ICV$.
4. Die Chiffre $C = (X||ICV) \oplus KS$ zusammen mit dem IV werden übertragen, ergo (C, IV).

Am Empfänger, wenn eine Chiffre zusammen mit dem IV, also (C, IV) eingeht, kommt es zu folgenden Aktivitäten:

1. IV und K werden zusammengeführt und es entsteht $S = IV||K$.
   Es wird der Schlüsselstrom erzeugt: $KS := k_0, k_1, \ldots, k_{n-1} \leftarrow PRNG(S)$.
2. Der Empfänger erhält $X||ICV = KS \oplus C$ und de-konkateniert in X' und ICV'.
3. Es wird ICV durch eigene Berechnung geprüft: $ICV\ ? = CRC\ (X')$.

Der Vorgang der Verschlüsselung mit WEP sowie der empfängerseitigen Entschlüsselung ist noch einmal in Abb. 3.8 schematisch aufbereitet.

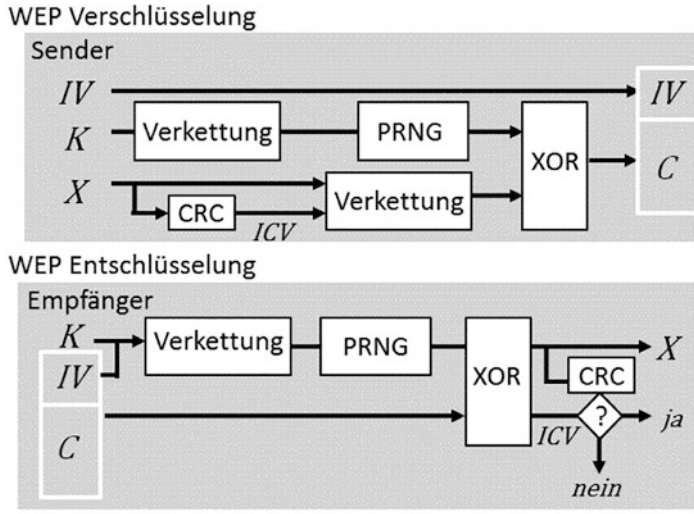

Abb. 3.8 Schematische Darstellung der WEP-Verschlüsselung und der WEP-Entschlüsselung bei mitsamt Prüfsummenbildung. (Quelle: eigene Darstellung)

3.5.1.1 Verwundbarkeit der Vertraulichkeit von WEP

Wir kennen die genauen Abwägungen in den Für- und Wider-Diskussionen aus derartigen Standardisierungssitzungen innerhalb der IEEE natürlich nicht, wissen jedoch, was das Ergebnis dieser Debatte war: Es wurden gerade einmal 24 Bit für den *IV* ‚spendiert'. Damit hat ein Angreifer leichtes Spiel. Denn schafft er es, den Access-Point unter Volllast zu setzen, würde es bei einem Durchsatz von nur 11 Mbit/s und ausgelastetem Access-Point nach einigen Stunden zu einer Wiederholung des ursprünglich gewählten Wertes für den *IV* führen. Fängt der Angreifer nun solch ein Paket ab, so ist dessen Chiffrat mit einem Schlüssel und einem *IV* erzeugt, die beide exakt in dieser Kombination schon einmal zur Verschlüsselung eines früher versendeten Pakets verwendet wurden. Es wurde also schon ein Paket unter Verwendung desselben Tupels (*IV*, *K*) erstellt und anschließend übertragen. Gelingt es dem Angreifer nun, diese beiden Chiffren C_1 und C_2 ausfindig zu machen, so wendet er

$$C_1 \oplus C_2$$

an. Das Ergebnis $M_1 \oplus M_2 \leftarrow C_1 \oplus C_2$ ist eine Bitfolge, die zwar immer noch kein Klartext ist, jedoch enthält der Ausdruck $M_1 \oplus M_2$ eben auch keinen Schlüssel mehr, sondern besteht allein aus den Klartexten M_1 und M_2 der beiden versendeten Pakete. Der Schlüssel hat sich aufgrund der Eigenschaften von $\oplus$ ‚rausgekürzt'.

Denn, wenn die beiden Chiffrate $C_1 = M_1 \oplus (K, IV)$ und $C_2 = M_2 \oplus (K, IV)$ vom Angreifer abgefangen wurden, dann addiert der Angreifer die verschlüsselten Nachrichten C_1 und C_2 bitweise auf, also

$$(M_1 \oplus (K, IV)) \oplus (M_2 \oplus (K, IV))$$

und erhält mit

$$M_1 \oplus M_2$$

einen Ausdruck aus zwei Klartexten, die mit XOR verknüpft sind, aber eben ohne Schlüssel.

Was dem Angreifer nun hilft, um von $M_1 \oplus M_2$ auf die Klartexte M_1 und M_2 zu folgern, sind Techniken der Häufigkeitsanalyse: So liegt beispielsweise die Häufigkeit des Auftretens einzelner Buchstaben in einem repräsentativen englischen Text vor [11, S. 247]. Während der häufigste Buchstabe ‚e' mit einer Wahrscheinlichkeit von 12,5 % vorkommt, tritt der am wenigsten häufige Buchstabe ‚x' mit einer Häufigkeit von 0,19 % auf. Ähnliche Häufigkeitsanalysen dürften natürlich auch für andere Sprachen vorliegen. Und weiter: Aus dem TCP/IP-Referenzmodell für eine strukturierte Datenübertragung wissen wir, dass die Dateneinheiten der unteren Schichten immer auch die Köpfe der Protokolle der höheren Schichten enthalten. Konkret bedeutet dies für WLAN mit WEP, dass in einem IP-Netz immer auch statische Informationen der IPv4- bzw. IPv6-Köpfe sowie der TCP- oder UDP-Köpfe in den Nachrichten M_1 und M_2 enthalten sind, die der Angreifer genau kennt und in vorteilhafter Weise zur Dekodierung von $M_1 \oplus M_2$ in M_1 und M_2 nutzen kann.

Der Grund für die Verwendung eines sich mit jedem zu versendenden Paket ändernden Initialisierungsvektors (*IV*) bei gleichzeitig statischem symmetrischem Schlüssel *K* liegt nun auf der Hand: Wohlwissend um die Schwäche von $\oplus$, welche sich bei seiner unsachgemäßen Nutzung ergibt, nämlich das ‚Rauskürzen' des Schlüssels aus einem Ausdruck $C_1 \oplus C_2$ mit gleichem Schlüssel *K* für C_1 und C_2, hoffte man durch den *IV* genügend Frische in das Konstrukt zu injizieren, um dieser Schwäche zu begegnen. Durch die sehr limitierte Größe für den *IV* war dies jedoch ein eher halbherziges Unterfangen.

Schlimmer noch: Einige Hersteller von Access-Points haben eine Zeit lang eine sehr fragwürdige Interpretation bei der Umsetzung des WEP-Standards betrieben. Der Standard verlangt für die Ausgestaltung des 24 Bit-IVs eine ‚zufällig' gewählte Bitfolge. Nun steckt in dem Wörtchen ‚zufällig', will man dies algorithmisch und digital umsetzen, schon so etwas wie die ‚Büchse der Pandora' wie viele der Leser wissen dürften. Ohne an dieser Stelle ins Detail gehen zu wollen, worin die Unterschiede von Zufallszahlengeneratoren wie TRNG, PRNG etc. liegen, so ist doch eines offensichtlich: das einfache Inkrementieren eines *IV*s immer dann, wenn ein weiterer *IV*-Wert benötigt wird, kann definitiv nicht die Lösung sein.

▶ Der Angriff auf die Verschlüsselung durch WEP macht sich zunutze, dass zwei Chiffrate C_1 und C_2, die mit dem gleichen Tupel aus Schlüsselstrom *K* und *IV* verschlüsselt wurden, mittels bitweiser XOR-Operation zu einem Ausdruck $C_1 \oplus C_2$ bearbeitet werden können, der keinen Schlüssel mehr enthält. Da bei WEP die Länge des Initialisierungsvektors *IV* auf 24 Bit begrenzt ist, gelingt das Auffinden solch eines Chiffrat-Paares C_1, C_2 recht zügig. Die Verwundbarkeit resultiert aus einer **fehlerhaften Spezifikation** und wurde bei der Umsetzung durch die teilweise sehr **laxe Implementierung des *IV*-Zählers** einiger Hersteller noch verstärkt.

Doch genau dies ist geschehen: Bei der Anmeldung eines Clients wird der *IV* initial auf 0 gesetzt (24 Nuller-Bits). Mit jedem weiteren Paket, das nun zwischen diesem Client und dem Access-Point übertragen wird, erhöht sich der neue Wert des *IV* jeweils um 1.

Für den Angreifer hat sich diese Umsetzung von ‚zufällig' als äußerst praktisch, weil zeitsparend erwiesen. Nun muss er den Access-Point nicht mal mehr unter Volllast setzen, um möglichst viele Pakete zu senden, sodass nach durchschnittlich $2^{24}/2$ generierter *IV*s er sicher sein kann, Paketpaare mit gleichem Schlüssel und gleichem *IV* abfangen zu können. Er muss also nur warten, bis sich ein weiterer Client am Access-Point anmeldet, denn dieser wird nun ebenfalls mit einem initialen Wert $IV = 0$ versorgt. Da der Schlüssel eh der gleiche ist, verwendet der Angreifer für $D = C_1 \oplus C_2$, also für die Chiffren C_1 und C_2 jeweils die ersten Pakete zweier unterschiedlicher Clients, die sich am Access-Point anmelden. Aus der Perspektive des Angreifers ist dies somit eine enorme zeitliche Vereinfachung des Angriffs auf eine mit WEP gesicherte WLAN Übertragung.

Abschließend sei noch angemerkt, dass zur Umsetzung des PRNG in WEP die Stromverschlüsselung RC4 verwendet wurde, von der wir schon seit längerer Zeit wissen, dass diese ebenfalls hochgradig unsicher ist.

Bekannte Angriffe der geschilderten Bauart auf WEP stammen aus 2001, der KoreK-Angriff aus 2004 sowie der PTW-Angriff [15] aus 2009. Bei den zuletzt genannten werden zur Sammlung der Daten ungefähr 60 min benötigt, wobei die eigentliche Berechnung dann in wenigen Sekunden durchführbar ist.

3.5.1.2 Verwundbarkeit der Integrität von WEP

Ähnlich verheerend war der Integritätsschutz von WEP, der das Einspielen unerkannter gefälschter Nachrichten unterbinden bzw. erkennbar machen sollte. Mit ein wenig Vorwissen über die Fähigkeiten einzelner kryptografischer Bausteine sollte recht schnell klar sein, dass ein Baustein zur zyklischen Redundanzprüfung in keinem Fall als ein Message Authentication Code zu verwenden ist, welcher die in Kap. 2 geschilderten Eigenschaften besitzt. Ein CRC ist einzig und allein geeignet, Bitfehler zu erkennen, die aufgrund von Rauschen auf der Übertragungsstrecke aufgetreten sind. Die augenscheinliche Schwäche bei der Verwendung zur Verhinderung von aktiven Angriffen liegt in seiner *Linearität,* also der Eigenschaft, die sich mathematisch wie folgt notieren lässt:

$$CRC(M_1 \oplus M_2) = CRC(M_1) \oplus CRC(M_2)$$

Verknüpft man nun also die CRCs zweier Klartexte M_1 und M_2 mittels XOR, so führt dies zum gleichen Ergebnis, als würde man zuerst beide Klartexte mit XOR verknüpfen und anschließend auf dem Ergebnis die Prüfsumme mittels CRC bilden.

So ist bei einer mit WEP gesicherten WLAN-Verbindung beispielsweise das empfängerseitig unerkannte Einspielen einer gefälschten verschlüsselten Nachricht C_2 möglich, sofern der Angreifer im Besitz eines Klartext-Krypto-Paares (M_1, C_1) ist. Da in die CRC-Berechnung selber keinerlei Schlüsselmaterial eingeht, funktioniert das Einschleusen eines Klartextes M_2 innerhalb eines verschlüsselten Bereiches C_2 wie folgt:

Nutze die Linearität zur Erlangung des aktuell verwendeten Schlüsselstroms K aus:

$$K \leftarrow C_1 \oplus (M_1 || CRC(M_1))$$

Berechne $CRC(M_2)$ und wende den oben erlangten Schlüsselstrom K an, sodass

$$C_2 \leftarrow (M_2 || CRC(M_2) \oplus K)$$

Bitte beachten Sie, dass der Angreifer hierfür nicht das zwischen WLAN-Access-Point und WLAN-Client ausgehandelte Geheimnis *Pairwise Master Key* (PMK) kennen muss. Der Angreifer kann den aktuell verwendeten Schlüsselstrom K selbst berechnen!

Nun könnte man argumentieren, dass es für einen Angreifer gar nicht so einfach ist in den Besitz eines Klartext-Krypto-Paares (M_1, C_1) zu gelangen. Doch dies ist mitnichten der Fall: Durch die Verwendung einer *Challenge-Response-Authentifizierung* in WEP kommen diese Informationen quasi frei Haus zum Angreifer: Denn zur Authentifizierung

sendet der Client als *Challenge* einen Zufallswert *RAND* an den Access-Point den dieser als *Response* mittels $E_{PMK}(RAND)$ beantwortet. Der Client entschlüsselt nun den vom Access-Point erhaltenen Chiffre-Text $E_{PMK}(RAND)$, also

$$RAND \leftarrow D_{PMK}(E_{PMK}(RAND))$$

Stimmen RAND′ und RAND überein, so hat der Client sich vergewissert, dass er sich bei einem Access-Point eingewählt hat, der über den gleichen *PMK* verfügt wie er selbst.

Ganz nebenbei hat ein Angreifer, der sich während dieser *Challenge-Response-Authentifizierung* in der Übertragungsreichweite befindet und die Daten mitgeschnitten hat, auch das für seinen Angriff erforderliche Klartext-Krypto-Paar (M_1, C_1) erhalten:

$$(M_1, C_1) := (RAND, E_{PMK}(RAND))$$

Die Abfolge dieses Angriffs zum Einschleusen gefälschter Daten in eine mit WEP gesicherte WLAN-Verbindung ist Abb. 3.9 noch einmal zusammengefasst.

▶ Der Angriff auf die Integrität von WEP macht sich die Verwendung einer zyklischen Redundanzprüfung *CRC* zum Zwecke des Integritätsschutzes zunutze. Insbesondere die lineare Eigenschaft von *CRC* sowie die Tatsache, dass kein Schlüssel eingeht, ermöglichen dem Angreifer, eigene Daten in eine mit WEP geschützte Verbindung zu schleusen. Die Vorbedingung, dass der Angreifer im Besitz eines Klartext-Chiffrat-Paares sein muss, wird durch WEP selbst unterstützt, indem die Client-Authentifizierung mittels Challenge-Response erfolgt: Die Verwundbarkeit resultiert aus einer **fehlerhaften Spezifikation.**

Obige Bemerkungen zur Verwundbarkeit der Vertraulichkeit und der Integrität unter der Verwendung von WEP dokumentieren, warum das *Bundesamt für Sicherheit in der*

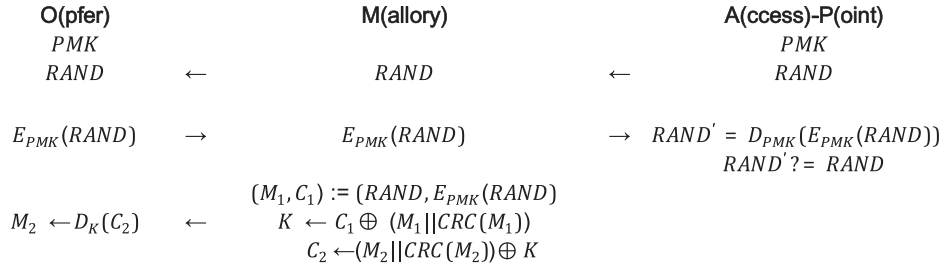

Abb. 3.9 Sniffen eines Klartext-Krypto-Paares (M_1, C_1) während der Challenge-Response-basierten Authentifizierung des Clients gegenüber dem Access-Point. Das Paar (M_1, C_1) dient der Vorbereitung des Einschleusens gefälschter Nachrichten M_2 an den Client. (Quelle: eigene Darstellung)

Informationstechnik (BSI) in seiner Bewertung zu WEP dieses in allen Kategorien mit ungenügend einstuft. Dies betrifft sowohl die Authentifizierung, die Integritätsprüfung als auch die Verschlüsselung.

3.5.2 WiFi Protected Access und IEEE 802.11i

Mit WiFi Protected Access (WPA) aus dem Jahre 2003 und IEEE 802.11i (WPA2) aus dem Jahre 2004 sind dann gleich zwei WLAN-Sicherheitserweiterungen entwickelt und verabschiedet worden. Denn eines der zu meisternden Probleme war es, eine Sicherheitserweiterung anzubieten, welche abwärtskompatibel ist, also auch zusammen mit sich noch massenhaft im Einsatz befindenden Geräten mit WLAN/WEP funktioniert. Gleichzeitig wurde zu dieser Zeit im Jahre 2000 auch die damals neue entwickelte und aus der Blockchiffre Rijndael hervorgegangene Blockchiffre Advanced Encryption Standard (AES) als dringend notwendiger Ersatz für den doch bereits arg in die Jahre gekommenen Data Encryption Standard (DES) von der NIST verabschiedet. Diese Ausgangslage führte dann zu zwei Weiterentwicklungen:

1. *WiFi Proteced Access (WPA),* als abwärtskompatible Variante zu WEP, bei der maximal ein Update der Firmware erforderlich sein durfte, sowie
2. *IEEE 802.11i (WPA2),* mit Verwendung von AES als nicht mehr abwärtskompatible Variante zu WEP, dafür aber mit Verwendung eines Krypto-Bausteins auf dem Stand der Technik zur Gewährleistung des besten augenblicklich verfügbaren Sicherheitsniveaus.

3.5.2.1 ARP – Spoofing im WLAN

Wie wird man eigentlich ein passiver Angreifer Eve mit Proxy-Funktionalität im eigenen WLAN, sodass der Verkehr eines weiteren Clients über das eigene Angreifer-Gerät geleitet wird, bevor er an den eigentlichen Access-Point geht? Eine Möglichkeit ist es, hierzu das *Address Resolution Protocol* (ARP) auszunutzen. Bei diesem Protokoll, das auf den *Request for Comment* (RFC) 826 der IETF [12] aus dem Jahre 1982 zurückgeht, fehlt jegliche Authentifizierung der versendeten Broadcast-Rahmen. Damit wird es einem Angreifer sehr leicht gemacht, derartige Rahmen zu seinen Gunsten zu verfälschen. Generell dient ARP dazu, dass IP-Verkehr innerhalb eines LAN an das richtige Gerät geleitet wird. Hierzu versorgt ARP alle Geräte im (W)LAN mit Informationen der jeweiligen *Medium-Access-Control* -(MAC-)-Adresse zur IP-Adresse. Möchte ein Gerät A nun die MAC-Adresse MAC_B eines anderen Gerätes B mit bekannter IP-Adresse IP_B in Erfahrung bringen, so setzt es einen ARP-Request der Form

$$ARP - Req[bc](MAC_A, IP_A, ?, IP_B)$$

ab. Da der ARP-Request die Broadcast-Adresse $bc := ff : ff : ff : ff : ff : ff$ enthält, wird diese Anfrage von allen Geräten im (W)LAN empfangen und geprüft. Das Gerät

B erkennt seine IP-Adresse in der Anfrage und antwortet mit einem ARP-Response der Bauart

$$ARP - Res[A](MAC_B, IP_B, MAC_A, IP_A)$$

indem es das Gerät *A* nun direkt per Unicast adressiert. Das Gerät *A* hat auf diese Weise die MAC-Adresse des Gerätes *B* erfahren und legt das Tupel (MAC_B, IP_B) in seinem lokalen ARP-Cache für zukünftige Verwendung für das Weiterleiten und Adressieren von IP-Verkehr ab.

Mit dem Wissen um das Protokollverhalten von ARP kann ein Angreifer vom Typ Insider nun zum Proxy zwischen einem beliebigen Client im WLAN und dem Access-Point werden, selbst dann, wenn dieses WLAN mit WPA oder WPA2 gesichert ist. Ähnlich wie das ARP-Poisoning zum Beispiel mittels des Werkzeuges Ettercap im drahtgebundenen *Local Area Network* (LAN) den ARP-Cache anderer Clients manipuliert und somit den Verkehr zu anderen Rechnern an sich umleitet, so gilt dies auch für WLAN-Netze.

Je nachdem, ob die *optionale* Erweiterung IEEE 802.11w zum Einsatz kommt oder nicht, erfolgt der Einsatz eines *Group Transient Key (GTK),* manchmal Group Temporal Key genannt, zur Absicherung einzelner per Broadcast gesendeter Management-Rahmen. Und dies auch nur dann, wenn derartige Rahmen nicht schon vor dem ersten Vier-Wege-Handshake benötigt werden. Dies führt dazu, dass der *GTK* in einigen Konstellationen tatsächlich vorhanden und ausgehandelt wurde, oftmals jedoch kein Gruppenschlüssel innerhalb eines WLAN zur Absicherung derartiger Broadcast-Nachrichten verwendet wird. Ist der Schlüssel nicht vorhanden, so braucht unser Angreifer Mallory sich nur in der Übertragungsreichweite zu befinden. Wurde ein *GTK* verwendet, so muss er darüber hinaus ein Insider sein. Denn um hier Proxy werden zu können, muss der Angreifer im Besitz des WLAN-Gruppenschlüssels, des Group Transient Key *(GTK)* sein, mit dem der Broadcast-Verkehr gesichert ist. Das eigentliche ARP-Spoofing erfolgt dann in beiden Fällen gleich.

Mallory versendet manipulierte ARP-Rahmen, um den Cache des Opfergerätes so zu verändern, dass dieser einen anderen Access-Point als den ursprünglichen in seinem ARP-Cache führt. Vergiftet Mallory auf die gleiche Weise auch noch den Cache des Access-Points, so werden über ihn alle Nachrichten in Richtung Opfer zum Access-Point sowie in umgekehrter Richtung gesendet. Die Abb. 3.10 verdeutlicht die Abfolge solch eines ARP-Poisoning-Angriffs im WLAN. Eine mit dem Gruppenschlüssel *GTK* (bzw. $GTK = null$) gesicherte ARP-Anfrage ($ARP - Req$) nach der MAC-Adresse MAC_{BS} zur IP-Adresse IP_{BS} wird vom Opfergerät per Broadcast gesendet. Da Mallory ebenfalls den Schlüssel *GTK* hält bzw. in den meisten Fällen, wie oben dargelegt, gar nicht benötigt, kann er eine ARP-Antwort ($ARP - Res$) absetzen, in der er seine eigene MAC-Adresse nennt und das Opfergerät so glauben machen kann, diese Adresse wäre die zur IP-Adresse der Internetschicht der Basisstation korrespondierende MAC-Adresse der Netzzugangsschicht. Mit Erhalt einer derartigen ARP-Res-Nachricht gelangen die vergifteten Daten (MAC_M, IP_{BS}) in den ARP-Cache des Opfergerätes. Auf die gleiche Weise kann Mallory auch den Cache

	O(pfer)		M(allory)$_{(Insider)}$		A(ccess)-P(oint)
1.	$ARP - Req_{GTK}$	→	$ARP - Req_{GTK}$	→	$ARP - Req_{GTK}$
	$[bc](MAC_O, IP_O, 0, IP_{BS})$		$[bc](MAC_O, IP_O, 0, IP_{BS})$		$[bc](MAC_O, IP_O, 0, IP_{BS})$
2.	Cache := (MAC_M, IP_{BS})	←	$ARP - Res_{GTK}$		
			$[O](MAC_M, IP_M, MAC_M, IP_{BS})$		
3.	$ARP - Req_{GTK}$	←	$ARP - Req_{GTK}$	←	$ARP - Req_{GTK}$
	$[bc](MAC_{BS}, IP_{BS}, 0, IP_O)$		$[bc](MAC_{BS}, IP_{BS}, 0, IP_O$		$[bc](MAC_{BS}, IP_{BS}, 0, IP_O)$
4.			$ARP - Res_{GTK}$	→	Cache := (MAC_M, IP_O)
			$[M](MAC_M, IP_M, MAC_M, IP_O)$		
			P(roxy)$_{Insider}$		
5.	a	→	a	→	a
6.	b	←	b	←	b

Abb. 3.10 WLAN-ARP-Spoofing eines bereits angemeldeten ‚Insider Mallory', um unerkannt Proxy im WLAN-Netzwerk zu werden. (Quelle: eigene Darstellung)

der Basisstation mit (MAC_M, IP_O) vergiften und ist von diesem Zeitpunkt an Proxy für jegliche Nachrichten zwischen Opfergerät und Access-Point.

Als Werkzeug für diesen Angriff hat sich neben arpspoof auch Ettercap unter der Lizenz GPLv2 für die Plattformen Linux und macOS etabliert. Im grafischen Modus wird es aus der Shell mit `ettercap -G` [4] gestartet. Als Alternative ist in [1] ein *Proof of Concept* (PoC) für ARP-Spoofing in Python unter Verwendung der Python-Bibliothek scapy [14, 20] beschrieben. Dieser enthält neben einer Sniffer-Funktion hauptsächlich eine Spoofing- und eine Restore-Funktion. Die Funktion namens Spoof entspricht im Wesentlichen dem Versenden der Nachrichten 2 und 4 in Abb. 3.10. Das Setzen von `op=2` bedeutet, dass es sich hier um einen ARP-Response handelt.

```
def Spoof(routerIP, victimIP):
    victimMAC=MACsnag(victimIP)
    routerMAC=MACsnag(routerIP)

    send(ARP(op=2, pdst=victimIP, psrc=routerIP, hwdst=victimMAC))
    send(ARP(op=2, pdst=routerIP, psrc=victimIP, hwdst=routerMAC))
```

Restore sendet dann mittels Broadcast-Nachrichten einen ARP-Response an das Opfer und den Access-Point. Danach ist die Proxy-Verbindung etabliert. Um sicherzustellen, dass der Angriff auch tatsächlich erfolgreich durchgeführt wird, wird die Spoof()-Funktion wiederholt aufgerufen bevor die Restore()-Funktion gestartet wird. Allerdings muss Mallory vorab das Weiterleiten von IP-Verkehr aktivieren. Benutzt er ein unixoides

System so geschieht dies durch die Ausführung von `echo "1" > /proc/sys/net/ipv4/ip_forward`.

```
def Restore(routerIP, victimIP):
    victimMAC=MACsnag(victimIP)
    routerMAC=MACsnag(routerIP)
    send(ARP(op=2, pdst=routerIP, psrc=victimIP,
    hwdst=''ff:ff:ff:ff:ff:ff'', hwsrc=
         victimMAC), count=4)
    send(ARP(op=2, pdst=victimIP, psrc=routerIP,
    hwdst=''ff:ff:ff:ff:ff:ff'', hwsrc =
         routerMAC), count=4)
```

Damit der Angriff erfolgreich sein kann, muss Mallory sich natürlich zeitgleich sowohl in der Übertragungsreichweite des Opfergerätes als auch in der Reichweite der Basisstation befinden. Ist dies der Fall, so kann auch die Verwendung von WPA oder WPA2 keine Abhilfe schaffen. Welche Auswirkung die Verwendung von WPA3 auf WLAN-ARP-Spoofing hat, werden wir dann in Abschn. 3.5.4 gesondert erörtern.

Übrigens: Wer nun unsicher ist, ob sein (W)LAN-fähiges Gerät auch einen vergifteten ARP-Cache nutzt, kann dies über das Shell-Kommando `arp -a` leicht prüfen. Mit Setzen der Optionen `-d` und `-s` können mit Systemrechten dann bedarfsweise Einträge im eigenen Cache gelöscht bzw. hinzugefügt werden. Wer den ARP-Verkehr im lokalen Netz aufzeichnen möchte, der kann dies darüber hinaus mit dem Werkzeug arpwatch unter Angabe der jeweiligen Schnittstelle durchführen und zwar mittels des Aufrufs `arpwatch -i wlan0`. Eine ‚flip flop'-Meldung im Systemlog nach Aufruf von arpwatch offenbart hierbei beispielsweise den häufigen Wechsel einer MAC-Adresse zu einer IP-Adresse und deutet somit u.U. auf einen ARP-Spoofing-Angriff hin. Weitere Meldungen wie ‚new activity', ‚new station', oder ‚changed ethernet address' deuten ebenfalls auf Unregelmäßigkeiten hin.

- Der ARP-Spoofing-Angriff auf WLANs mit WPA oder WPA2 als Sicherung und einem Insider Mallory zielt darauf ab, Proxy zwischen Opfergerät und Access-Point zu werden. Der Angriff macht sich zunutze, dass sowohl ARP-Anfragen als auch ARP-Antworten nur mittels des WPA/WPA2-Gruppenschlüssels *GTK* gesichert sind. Jedes Gerät im Netzwerk, das in Besitz dieses Schlüssels ist, kann somit den ARP-Cache eines anfragenden Gerätes mit manipulierten Einträgen vergiften und z. B. vorgaukeln, die Basisstation zu sein. Die Verwundbarkeit resultiert aus einer **fehlerhaften Spezifikation** von ARP gegenüber manipulierten ARP-Antworten.

Anbei noch einige Anmerkungen zu der bisher optionalen Erweiterung 802.11w: Der Gruppenschlüssel *GTK* ist erst nach dem Vier-Wege-Handshake verfügbar, da er dem

Client hierüber mitgeteilt wird. Der *GTK* wird ausschließlich für Integritätsschutz und Authentifizierung von Broadcast-Nachrichten im WLAN verwendet. Danach sind eine Reihe von Broadcast-Nachrichten beispielsweise zur Initiierung eines Verbindungsabbaus, zur nochmaligen Authentifizierung sowie zum robusten Management hierüber abgesichert. Nun ist es aber so, dass ARP-Nachrichten aus Sicht des WLAN-Standards ‚Daten' sind. Für den Fall, dass derartige lokal per Broadcast gesendete Daten ebenfalls mit *GTK* gesichert sind, ist das ARP-Spoofing im WLAN nur als Insider möglich, ansonsten auch als externer Angreifer.

Der unter *Hole 196* bekannt gewordene Angriff ist schon seit 2010 öffentlich. Dieser Angriff bezieht seinen Namen aus derjenigen Seite im IEEE 802.11-Standard aus dem Jahre 2007, von der aus die Verwundbarkeit für die Sicherheitsexperten ableitbar war. Bei diesem Angriff wird ausgenutzt, dass mit dem *GTK* gesicherte Rahmen, im Gegensatz zu den mit dem *PTK* (Pairwise Transient Key) gesicherten Rahmen, nicht auf ihre MAC-Adressen überprüft werden. Damit ist es aber möglich, dass neben dem Access-Point auch hierfür unbefugte Geräte mit Kenntnis des Gruppenschlüssels *GTK* derartige Rahmen versenden können. Neben dem Versenden von beispielsweise schadhaftem Code an einen Client kann mittels Hole 196 eben auch eine Manipulation von ARP-Tabellen einzelner Clients vorgenommen werden um Proxy-Insider zu werden und dies wohlgemerkt auch unter Verwendung von WPA2. Weiterführende Details zu Man-in-the-Middle-Angriffen auch auf mit WPA2 verschlüsselte WLANs sind in der Arbeit [1] von Agrarwal und Co-Autoren geschildert.

Zum Abschluss noch eine Anmerkung zum sogenannten *Promiscuous*-Modus bei WLAN. Wird dieser Modus verwendet, so liest das Gerät den gesamten an der Schnittstelle anfallenden Netzwerkverkehr mit und reicht eben nicht nur den für ihn bestimmten Verkehr an seine höheren Protokollschichten weiter. Dies scheint also der ideale Modus zur Umsetzung der Rolle E(ve) zu sein. Allerdings geht die Etablierung einer Verbindung mit einer Authentifizierung zwischen Client und Access-Point einher, sodass nach der Etablierung auch im *Promiscuous*-Modus der an der Schnittstelle anfallende Verkehr eben nicht allem Verkehr im WLAN entspricht.

3.5.2.2 WPA – WiFi Protected Access

Die wesentliche Neuerung bei WPA gegenüber WEP war das *Temporary Key Integrity Protocol* (TKIP), das den eingangs geschilderten Verwundbarkeiten von WEP durch Einführung frischer Schlüssel pro zu übertragenden Datenrahmen begegnet. Die einzelnen TKIP-Erweiterungen waren die Einführung eines *Message Integrity Code* (MIC) mittels eines „Michael" genannten Algorithmus zur Integritätsprüfung anstelle des ursprünglich hierzu verwendeten *Cyclic Redundancy Codes* (CRC), die Erweiterung des IV von 24 Bit auf 48 Bit, ein unterschiedlicher Schlüssel für jedes Datenpaket sowie die Etablierung eines Schlüsselerneuerungsprotokolls unter Beibehaltung der RC4-Komponente für den PRNG.

Zur Erzeugung eines jeweils frischen Paketschlüssels zum Verschlüsseln wird aus dem 256 Bit langen *Pairwise Master Key* (PMK) mittels Zufallszahlengenerator PRF-X

ein 512 Bit langer *Pairwise Transient Key* (PTK) abgeleitet, wobei neben zweimal 128 Bit für den Gruppenschlüssel zusätzlich 128 Bit als Temporal Key 1 (TK1) zum Verschlüsseln und weitere 128 Bit als Temporal Key 2 (TK2) zur Integritätssicherung mittels Michael dienen. Gemeinsam mit der MAC-Adresse des Senders und einem TKIP-Reihenfolgenzähler dient der TK1 der Erzeugung des aktuellen Paketschlüssels. Ähnlich erfolgt die Schlüsselerzeugung für die MIC-Berechnung: Der TK2 zusammen mit der MAC-Quell- und Zieladresse und dem TKIP-Reihenfolgezähler gehen in die MIC-Berechnung der MAC-Nutzdaten ein. Diese Maßnahmen zur Schlüsselerzeugung sind bei WPA der ursprünglichen Verschlüsselung mittels WEP vorgeschaltet.

Zu den Designanforderungen der Komponente Michael sind einige Anmerkungen angebracht: Um die Abwärtskompatibilität zu gewährleisten, sollte das Verfahren ohne weitere kryptografische Verfahren auskommen. Des Weiteren sollte der Mehraufwand am Access-Point so gering als möglich ausfallen. Die Lösung hierfür sah vor, dass zur Integritätsprüfung lediglich Shift-, Addition- und XOR-Operationen zu verwenden sind und insbesondere keinerlei Multiplikationen. All dies führte jedoch dazu, dass das Verfahren anfällig gegenüber Brute-Force-Angriffen sowie Angriffsformen basierend auf differentieller Krypto-Analyse wurde. Daher erfolgt eine Unterbindung des Netzverkehrs für eine Minute bei der Erkennung einer Unregelmäßigkeit. Ein eventueller Integritätsangriff wird somit unterbunden. Doch dies geschieht auf Kosten eines quasi implizit eingebauten DoS-Angriffes.

Natürlich hat eine erhöhte Sicherheit fast immer ihren Preis und im Falle von WPA sind dies insbesondere die Auswirkungen auf die Nettodatenrate auf dem drahtlosen Medium. Für MIC, ICV und nun längeren IV fallen insgesamt 12 Byte mehr Signaldaten pro zu übertragenden Rahmen an. Dies macht sich in einem Durchsatzverlust der Nutzdaten von 10–15 % gegenüber rein WEP gesichertem Verkehr bemerkbar.

3.5.2.3 WPA2 – IEEE 802.11i

Die zweite, nicht mehr abwärtskompatible Variante IEEE 802.11i [8] WPA2 aus dem Jahre 2004 nutzt, wie eingangs bereits gesagt, die Blockchiffre AES. Sie verwendet hierzu den Verkettungsmodus CTR mit CBC-MAC (CCMP), den *Counter Mode* unter Verwendung des *Cipher Block Chaining Message Authentication Code Protocol.* Ein Verkettungsmodus (mode-of-operation) dient dazu, die einzelnen 128 Bit-Chiffreblöcke so aneinander zu binden, dass nicht einzelne Chiffreblöcke austauschbar oder in ihrer Reigenfolge unerkannt veränderbar sind. Dabei erfolgt das Authentifizieren und Verschlüsseln der Daten nicht separat, sondern integriert in einer Abfolge. Vereinfacht gesagt wird AES in WPA2 zur Erzeugung temporärer Schlüssel pro 128 Bit-Block von zu übertragenden Nutzdaten eines MAC-Rahmens verwendet, also nicht zur eigentlichen Verschlüsselung der Daten. Diese geschieht nach wie vor mit dem gut verstandenen XOR-Baustein. Ein wenig vereinfacht dargestellt lässt sich sagen, dass eine Zählerinitialisierung auf Basis der Paketnummer und Informationen des MAC-Kopfes erfolgt. Diese Zählerinitialisierungen dienen als Zählerinput, die zusammen mit einem temporären Schlüssel die Eingabe für die AES-Verschlüsselung bilden. Das hieraus resultie-

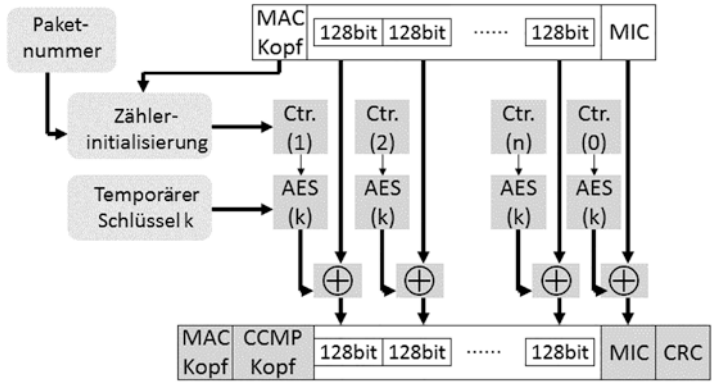

Abb. 3.11 Vereinfachte Darstellung der Verwendung von AES innerhalb von WPA2 zur gleichzeitigen Verschlüsselung und Authentifizierung. (Quelle: eigene Darstellung in Anlehnung an [8])

rende Chiffrat der AES-Verschlüsselung dient nun als der jeweilige temporäre Schlüssel für die eigentliche Verschlüsselung eines 128 Bit-Klartext-Blocks aus den Nutzdaten des MAC-Rahmens. Auf die gleiche Weise erfolgt abschließend die Bearbeitung des MIC. Ein zu versendender Rahmen enthält neben den derart verschlüsselten 128 Bit-Blöcken und MIC zwischen MAC-Kopf und Nutzdaten noch einen CCMP-Kopf und schließt mit einem CRC ab. Dieser dient ausschließlich der Behandlung von Übertragungsfehlern. Die Abb. 3.11 stellt die Verwendung von AES innerhalb von WPA2 noch einmal in einer schematisch vereinfachten Form dar.

Zusammenfassend können wir feststellen, dass WPA (siehe hierzu auch Abschn. 3.5.2.4) keine wesentlich höhere Sicherheit bietet als WEP. Das ist spätestens seit 2008 bekannt, als Erik Tews und Martin Beck [15] eine Entschlüsselung auf *ChopChop*-Art (Aufspalten) aufzeigten und sich dabei einige realistische Gegebenheiten zunutze machten: So sind für den Angreifer typischerweise die Adressbereiche von TKIP und IPv4 bekannt. Des Weiteren kann sich ein Angreifer ein relativ langes Intervall für das Re-Keying zunutze machen, das bei 3600 s liegt. Zudem kann der Angreifer verschlüsselte ARP-Anfragen und -Antworten mithören und er kennt, was nicht unwesentlich ist, die typische Länge von Nachrichten sowie oftmals einzelne Passagen aus den Klartexten. Diese zusätzlichen Informationen führen dazu, dass ein Angreifer, der eine gewisse Zeit Zugang zum Netzwerk hat TKIP, verschlüsselten Verkehr in ChopChop-Manier entschlüsseln kann, auch wenn TKIP anders als WEP schon zumindest ein wenig gegen derartige Angriffsformen gehärtet wurde.

▶ Der ChopChop-Angriff auf die Verschlüsselung durch WPA mittels TKIP macht sich zunutze, dass verschlüsselte ARP-Anfragen und -Antworten, sowie Informationen über die Länge von Nachrichten und deren Inhalte aufgrund fester Wertebereiche für IP-Adressen zu einer signifikanten Einschränkung des

Suchraumes führen. Die Verwundbarkeit resultiert aus dem **Ausnutzen von zusätzlichen Informationen durch andere an der Übertragung beteiligter Protokolle** wie ARP.

WPA2 dagegen galt bis zum Jahre 2017 als sicher. Doch mehr dazu in Abschn. 5.3.4.

3.5.2.4 Aus der Praxis

`Aircrack` und `Airodump` fangen WLAN-Datenpakete ab und können eigene Pakete auf der WLAN-Schnittstelle senden. So kann man sich mit `airodump-ng` alle verfügbaren Netzwerke anzeigen lassen. WPA kann nun angegriffen werden, wenn mindestens eine Station mit dem Netzwerk verbunden ist. Für den Angriff benötigt man einen abgefangenen WPA-Handshake. `Aireplay-ng` kann zwei Stationen zu einem erneuten Handshake zwingen. Gleichzeitig muss `airodump-ng` die Pakete aus der Luft mitlesen. Airodump muss permanent auf dem korrekten Kanal mithören und die abgefangenen Daten abspeichern, die dann mit dem Paket-Sniffer Wireshark angezeigt werden können. Nun kann man sich die vier „Extensible Authentication Protocol over Local Area Network"-(EAPOL-)Pakete des WLAN-Handshakes anzeigen lassen. Der Angreifer muss nun entscheiden, ob er mittels Aircrack eine einfache Wörterbuchattacke durchführen möchte oder unter Verwendung des Analysewerkzeuges cowpatty zunächst eine Rainbow-Tabelle anlegen will. Cowpatty kann mit dieser Tabelle sehr viel schneller nach dem richtigen Passwort suchen. Das Erstellen der Tabelle benötigt einige Zeit und ist nur für Netzwerke mit genau dieser BSSID (WPAsec) gültig.

3.5.3 Designschwächen von WPA und WPA2

Vielbeachtet war dann im Jahre 2017 die Bekanntmachung einer weiteren Verwundbarkeit der WLAN-Sicherheitslösungen, da diese neben WPA nun auch das bisher als sicher eingestufte WPA2 betraf. So haben die Sicherheitsexperten Mathy Vanhoef und Frank Piessens von der KU Leuven auf der ACM CCS'17 [16] einen Man-in-the-Middle Angriff-mit beachtlichen Auswirkungen aufgezeigt. Dieser Angriff zielt aber nicht auf das Aushebeln der verwendeten kryptografischen Bausteine ab. So ist die Verwendung des Advanced Encryption Standards (AES) im Verkettungsmodus CCMP-Mode zur Verknüpfung einzelner Chiffrat-Blöcke einer Blockchiffre, wie dies bei WPA2 der Fall ist, immer noch als berechenbar sicher einzustufen.

Allerdings war das Brechen der im WLAN verwendeten Krypto-Bausteine auch gar nicht das erklärte Ziel des von Vanhoef und Piessens aufgezeigten Angriffs. Vielmehr haben die Sicherheitsforscher einen Weg beschrieben, den Vier-Wege-Handshake, der im Rahmen des Aushandelns eines frischen Schlüssels benötigt wird, so zu beeinflussen, dass die für die Verschlüsselung verwendeten Schlüssel aufgedeckt bzw. wiederverwendet werden. Ähnliche Verwundbarkeiten konnten durch die Forscher auch für weitere

Handshake-Varianten gezeigt werden. So haben sie dies beispielsweise ebenfalls für den Prozess zum Aushandeln eines Gruppenschlüssels dargelegt.

Je nach dem verwendeten mobilen Betriebssystem hat dieser Angriff zum Teil verheerende Auswirkungen: So führt bei Android 6.0 auf der Seite des Clients dieser *als Key Reinstallation Attack* (KRACK) bezeichnete Man-in-the-Middle-Angriff auf den Vier-Wege-Handshake dazu, dass nach dem Angriff nur noch eine ‚All-zero'-Verschlüsselung verwendet wird, die Daten also völlig unverschlüsselt über die Luftschnittstelle übertragen werden. Eine ganze Reihe weiterer Betriebssysteme waren betroffen mit zum Teil ähnlich verheerenden Auswirkungen. Dies führte dazu, dass im Jahre 2017 ca. 1/3 aller Geräte verwundbar gegenüber diesem unter dem Namen KRACK bekannt gewordenen Angriff waren.

3.5.3.1 Grundidee des KRACK-Angriffs

Doch nun zur Grundidee des KRACK-Angriffs: Wenn ein Client eine gesicherte WLAN-Verbindung aufbauen möchte, so erfolgt dies über eine Assoziierungsphase. Diese Phase wird durch eine Nachricht des Clients an den Access-Point eingeleitet. Beim ersten Kontakt ist der Austausch dieser Assoziierungs- und Authentifizierung-Nachrichten offen und ungeschützt. An solch eine Assoziierungsphase schließt nun eine weitere Phase an. Diese Phase trägt den Namen Vier-Wege-Handshake. Sie besteht im Erfolgsfall aus dem Versenden von vier Nachrichten, deren Abfolge und Aufbau wir später noch ein wenig genauer betrachten werden. Dabei gehen die erste Nachricht und die dritte Nachricht vom Access-Point an den Client, die zweite Nachricht und die vierte Nachricht vom Client an den WLAN-Access-Point. Mit erfolgreicher Beendigung dieses Vier-Wege-Handshakes haben sich Client und Access-Point dann auf einen *frischen paarweisen Sitzungsschlüssel* verständigt. Dieser Schlüssel trägt den Namen *Pairwise Transient Key* (PTK). Der Client speichert diesen Schlüssel nach dem Eingang der dritten Nachricht. Ab diesem Moment verwendet er diesen Schlüssel zur Verschlüsselung aller weiteren Nachrichten, die im Zuge der drahtlosen Kommunikation anfallen.

Die prinzipielle Abfolge des Vier-Wege-Handshakes ist in Abb. 3.12 aufgezeigt. Ausgangspunkt hierbei ist der langlebige Pairwise Master Key (PMK), der beiden Parteien vorab bekannt sein muss. Der PMK lässt sich aus einem Hash über dem Namen des WLAN-Netzwerkes, also dem Service Set Identifier (SSID), und der Passphrase PSK berechnen:

$$PMK := h(SSID, PSK)$$

Der Access-Point erstellt nun eine Authenticator Nonce, notiert als N_A, und übermittelt diese in der ersten Nachricht des Vier-Wege-Handshakes zusammen mit der aktuellen Sequenznummer *sn* an den Client. Mit Erhalt der ersten Nachricht erzeugt der Client nun seinerseits eine Nonce (number used once) namens Supplicant Nonce N_S. Der Client berechnet darüber hinaus unter Verwendung eines Zufallszahlengenerators *PRNG*() den zeitlich begrenzten Pairwise Transient Key *(PTK).* In diese Berechnung gehen der

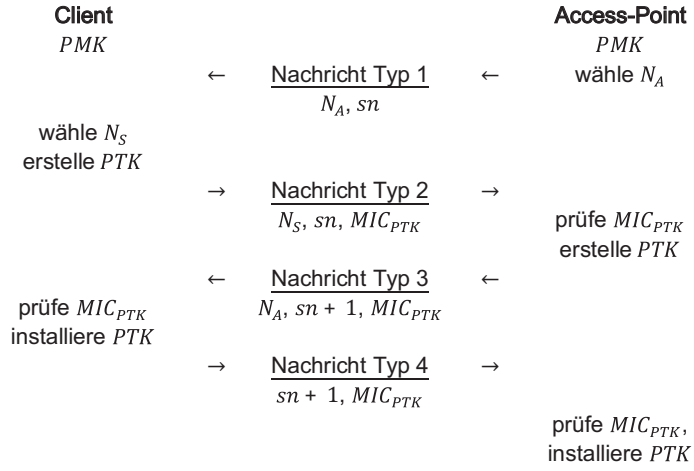

Abb. 3.12 Erfolgreiche Abfolge des Vier-Wege-Handshakes zwischen WLAN-Access-Point und Client. (Quelle: eigene Darstellung in Anlehnung an [16])

PMK, die Nonce N_A und N_S, die MAC-Adresse des Access-Points MAC_{AP} sowie die MAC-Adresse des Clients MAC_{Client} ein:

$$PTK := PRNG(PMK, N_A, N_S, MAC_{AP}, MAC_{Client})$$

Neben der N_S und der Sequenznummer *sn* antwortet der Client mit Nachricht 2 durch zusätzliches Einfügen eines Message Integrity Codes *(MIC)* unter Verwendung des *PTK*. Mit Erhalt dieser Nachricht ist der Access-Point seinerseits befähigt, den *PTK* zu berechnen und eine Prüfung des MIC durchzuführen, bevor er die Nachricht 3 an den Client sendet. Eine Nachricht vom Typ 3 enthält wiederum die N_A, den *MIC* sowie die um den Wert 1 inkrementierte Sequenznummer. Mit Erhalt der Nachricht 3 prüft der Client den *MIC* und installiert den *PTK* falls die Prüfung des *MIC* erfolgreich war. Ist dies erfolgt, sendet der Client eine Nachricht vom Typ 4, die wiederum die um 1 erhöhte Sequenznummer enthält sowie den *MIC*. Der Vier-Wege-Handshake ist beendet, nachdem der Access-Point die Nachricht 4 erhalten hat und daran anschließend seinerseits den *PTK* installiert hat.

Allerdings sieht die Abfolge des Vier-Wege-Handshake-Protokolls vor, dass der Access-Point in einigen Situationen das Übertragen der dritten Nachricht wiederholt. Solch ein Verhalten des Access-Points erfolgt immer genau dann, wenn der Access-Point nicht in einer vorab festgelegten Zeitspanne eine Bestätigung vom Client für den tatsächlichen Erhalt der dritten Nachricht bekommen hat. Oder, um im korrekten Protokoll-Jargon zu bleiben: Innerhalb der vom *Timer* vorgesehenen Zeitspanne ist kein *ACK* eingegangen. Dieses geschilderte Verhalten des WLAN-Access-Points ist nicht zufällig gewählt. Vielmehr erfolgt hier eine für die meisten Konstellationen praxistaugliche

Umsetzung, um den Vier-Wege-Handshake hinsichtlich des in Kap. 2 dieses Buches geschilderten Zwei-Armeen-Problems robust und fehlertolerant zu gestalten. Denn der Access-Point kann bei einer fehlerhaften Übertragung und dem nicht Erhalten der Bestätigungsnachricht niemals entscheiden, ob es zu einem Übertragungsfehler beim Versenden der eigentlichen Nachricht oder beim Versenden der Bestätigung zum Erhalt der Nachricht gekommen ist.

So kann es in einigen Konstellationen jedoch trotz der Verwendung von *Timern* immer noch vorkommen, dass aufgrund von Rahmenverlusten auf dem drahtlosen Medium der Client sehr wohl die dritte Nachricht erhalten hat, aber die Bestätigung, also das ACK, an den Access-Point verloren gegangen ist. Wie bereits geschildert, kann der Access-Point dies jedoch nicht unterscheiden und sendet somit ein zweites Mal die dritte Nachricht, mit der Konsequenz, dass beim Client zwei Nachrichten vom Typ 3 eingehen.

Welchen Effekt hat nun das mehrmalige Empfangen einer Nachricht vom Typ 3 auf der Seite des Clients? Hierzu muss man wissen, dass mit jedem Erhalt einer Nachricht vom Typ 3 der Client Folgendes in die Wege leitet:

1. Es erfolgt eine Re-Installierung des (gleichen) Sitzungsschlüssels.
2. Es erfolgt ein Zurücksetzen der inkrementellen Übertragungsnummer von Paketen *(Nonce).*
3. Empfang eines Replay-Zählers, der vom Verschlüsselungsprotokoll verwendet wird.

Mit diesem Wissen um das Verhalten des Clients bei Empfang einer Nachricht vom Typ 3 kann ein Angreifer nun versuchen, die *Nonce* zurückzusetzen, indem er Nachrichten vom Typ 3, die der Access-Point gesendet hat, speichert, um sie später noch einmal zu senden.

Die Abb. 3.13 zeigt eine sehr stark vereinfachte Darstellung des (informellen) Zustandsautomaten für den WLAN-Client in Anlehnung an die Beschreibung des Standards IEEE 802.11r [9]. Dabei geben die Kantenbeschriftungen Auskunft darüber, welche Ereignisse den Client von einem Zustand in einen anderen Zustand übergehen lassen. Wesentlich für das Verständnis des KRACK-Angriffs ist hierbei die Tatsache, dass der clientseitige Zustandsautomat das nochmalige Senden von Nachrichten des Typs 1 oder des Typs 3 explizit vorsieht. Dies erfolgt immer genau dann, wenn die Gegenstelle eine Nachricht vom Typ 2 bzw. vom Typ 4 innerhalb einer gewissen Zeitspanne nicht erhalten hat. Konkret bedeutet dies, dass der Client aus zwei unterschiedlichen Zuständen in den Zustand PTK-Aushandlung überführt werden kann:

1. Mit Erhalt einer Nachricht vom Typ 3 aus dem Zustand *PTK-Start* in den Zustand *PTK-Aushandlung*
2. Mit Erhalt einer Nachricht vom Typ 3 aus dem Zustand *PTK-Beendet* (zurück) in den Zustand *PTK-Aushandlung.*

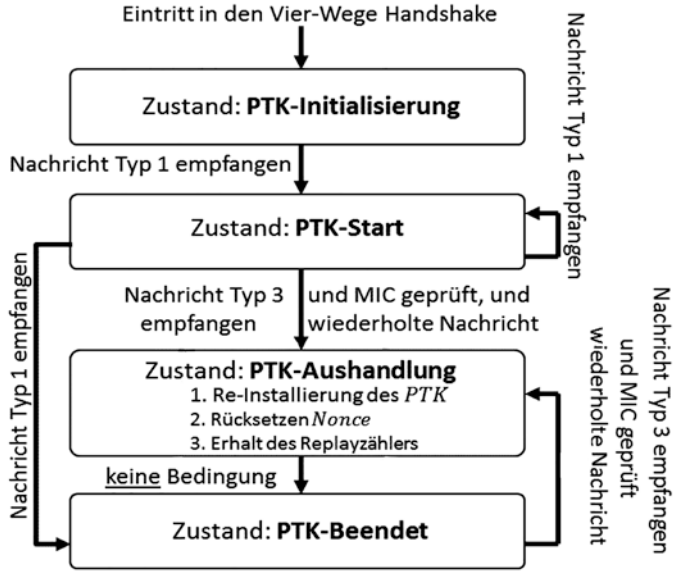

Abb. 3.13 Zustände und Zustandsübergänge am WLAN-Client in Anlehnung an IEEE 802.11.r. (Quelle: eigene Darstellung in Anlehnung an [16])

Welche weiterführenden Auswirkungen dies in Gegenwart eines findigen Angreifers Mallory auf die Absicherung der drahtlosen Übertragungsstrecke haben kann, werden wir nun in den nachfolgenden Abschnitten erörtern.

▶ Der KRACK-Angriff beruht damit im Kern auf dem *Zwei-Armeen-Problem* als einem bekannten allgemeinen Dilemma beim Entwurf von Kommunikationsprotokollen. Da es bei Ausbleiben einer Bestätigung für den Sender (Access-Point) unentscheidbar ist, ob die Gegenstelle (der WLAN-Client) die eigentliche Nachricht bereits erhalten hat, ermöglicht eine spezifikationskonforme Umsetzung, Nachrichten ein weiteres Male zu versenden. Diese werden auf der Clientseite dann nochmalig bearbeitet, ungeachtet des konkreten Zustandes *(PTK-Aushandlung, PTK-Beendet),* in dem sich der Client gerade befindet. Allgemein offenbart diese Schwäche bei der Ausgestaltung des Vier-Wege-Handshakes eine inhärente Abhängigkeit bei der Absicherung von Kommunikationsprotokollen von Konzepten eben dieser Kommunikationsprotokolle selbst. Die Verwundbarkeit durch den KRACK-Angriff resultiert aus einem **fehlerhaften Zusammenspiel von WPA/WPA2-Sicherheitsstandard und dem eigentlichen Kommunikationsprotokoll.**

3.5.3.2 Auswirkung des KRACK-Angriffs

Die konkreten Auswirkungen, welche sich aus dem KRACK-Angriff tatsächlich ergeben, hängen stark von dem verwendeten Protokoll zur Verschlüsselung der Daten ab. Mathy Vanhoef und Frank Piessens geben in [16] hierzu die folgenden Informationen. Für den Fall, dass *AES* im *CCMP*-Modus eingesetzt wird, was bei WPA2 zutreffen dürfte, gilt:

1. Zum einen können willkürliche Pakete entschlüsselt werden.
2. Außerdem ist es möglich, TCP-SYN-Pakete zu korrumpieren und dadurch ganze TCP-Verbindungen zu übernehmen. Die ermöglicht es beispielsweise, bösartige Inhalte in http-Verbindungen zu schleusen.

Für den Fall, dass *AES* mit *TKIP* oder aber *GCMP* eingesetzt wird, was bei Verwendung von WPA zutrifft, gilt:

1. der Angreifer kann willkürliche Pakete Entschlüsseln sowie seine eigenen Pakete injizieren;
2. Für den Fall, dass der Handshake zur Aushandlung des Gruppenschlüssels angegriffen wurde, kann der Angreifer Broadcast- und Multicast-Nachrichten wiederholt einspielen. Dieser Fall wird im Rahmen dieses Buches nicht weiter erörtert.

Die Auswirkungen des KRACK-Angriffs sind insbesondere besorgniserregend für Geräte unter Linux, die WPA_SUPPLICANT (Versionen 2.4 und 2.5) verwenden. Denn hier wird anstelle einer Re-Installierung eines echten Schlüssels, ein 0er-Schlüssel installiert. Dies ist der Fall bei Android 6.0 oder Android Wear 2.0, sodass im Jahre 2017 insgesamt mehr als 30 % aller Android-Geräte von der geschilderten Verwundbarkeit betroffen waren.

Allerdings gilt der IEEE 802.11i-Zusatz [8], in dem der Vier-Wege-Handshake spezifiziert ist, sowohl für WPA als auch für WPA2. Damit sind alle WiFi-Netze mit mehr oder weniger starken Auswirkungen vom KRACK-Angriff betroffen.

3.5.3.3 Praktische Aspekte bei der Umsetzung von KRACK

Beschäftigt man sich mit diesem Angriff ein wenig genauer, so wird doch recht schnell deutlich, dass dieser als MitM klassifizierte Angriff für Mallory doch ein wenig aufwendiger ist, als es nach der ersten Lektüre von Abschn. 3.5.3 erscheinen mag.

Zum einen lohnt sich eine genauere Betrachtung der erforderlichen Voraussetzungen für diesen MitM-Angriff. Zum anderen stellt sich heraus, dass einige Implementierungen die wiederholt einzuschleusende dritte Nachricht als Klartext-Nachricht akzeptieren, andere Implementierungen jedoch ausschließlich eine verschlüsselte dritte Nachricht in der Abfolge des Vier-Wege-Handshakes akzeptieren. Und schließlich zeigt sich, dass nicht alle Implementierungen eine saubere Umsetzung des Zustandsautomaten vorgenommen

haben. So erlauben beispielsweise die Betriebssysteme Windows und iOS kein wiederholtes Senden der dritten Nachricht. Sinnigerweise macht sie diese nicht standardkonforme Umsetzung immun gegen den KRACK-Angriff auf das Vier-Wege-Protokoll für das Aushandeln eines paarweisen symmetrischen Schlüssels. Geräte, auf denen diese Betriebssysteme laufen, bleiben jedoch immer noch anfällig gegen die in diesem Buch nicht behandelten Angriffe zur Erlangung des Gruppenschlüssels.

So wäre ein einfacher MitM-Angriff mittels eines ‚rohen' bzw. unter der Verwendung eines technisch unausgereiften WLAN-Access-Points nicht möglich, da dieser eine andere MAC-Adresse aufweisen würde als der Access-Point, dessen drahtlose Verbindung korrumpiert werden soll. Dies kommt insbesondere daher, dass die MAC-Adressen von Client und Access-Point in die Berechnung des gemeinsamen Sitzungsschlüssels eingehen, und somit unter Verwendung eines rohen WLAN-Access-Points der Client einen anderen Sitzungsschlüssel generieren würde als der Access-Point, dessen Verbindung korrumpiert werden soll. Ein weiteres Erfordernis neben gleicher MAC-Adressen besteht darin, dass der geklonte WLAN-Access-Point auf einem anderen Kanal senden muss. Ansonsten wäre ebenfalls die einheitliche Berechnung aufseiten des Clients und des Access-Points nicht gewährleistet. Details hierzu finden sich in [17].

Welche Systeme das Einschleusen einer unverschlüsselten dritten Nachricht tolerieren und welche Systeme das ausschließliche Einschleusen einer wiederholten verschlüsselten dritten Nachricht erwarten, ist abhängig von der konkreten Kombination aus verwendetem mobilem Betriebssystem und der konkret verwendeten drahtlosen Netzwerkschnittstellenkarte WiFI-NIC. Beide Fälle wollen wir nachfolgend zumindest in ihren Grundzügen behandeln.

3.5.3.4 Unverschlüsseltes wiederholtes Senden

Das wiederholte Senden einer Handshake-Nachricht ist relativ einfach, wenn man über einen MitM-Access-Point mit den oben geschilderten Eigenschaften verfügt. Das unverschlüsselte wiederholte Senden erfolgt in vier Phasen:

Phase 1 Der MitM-Access-Point blockiert die vierte Handshake-Nachricht, die der Client an den Access-Point sendet.

Phase 2 Der Opfer-Client installiert die Schlüssel *Pairwise Transient Key* (PTK) und *Groupwise Transient Key* (GTK), so wie es der IEEE 802.11i-Standard nach Erhalt der dritten Nachricht vom Access-Point vorsieht. Des Weiteren öffnet der Opfer-Client nun den 802.1x-Port zum Übertragen gewöhnlicher Rahmen.

Phase 3 Aufgrund der nicht erhaltenen vierten Nachricht sendet der zu korrumpierende Access-Point die dritte Nachricht nach Ablauf seines Timers *timer* ein weiteres Mal. Der MitM-Access-Point leitet diese einfach weiter, ohne Änderungen daran vorzunehmen. Das Eintreffen einer weiteren dritten Handshake-Nachricht veranlasst den Opfer-Client nun

- seine eigentliche vierte Nachricht zu blockieren,
- die Schlüssel *PTK* und *GTK* noch einmal zu installieren, sowie
- die Werte für Nonce und Replay-Zähler zurückzusetzen.

Konkret bedeutet dies, dass von nun an die zu verschlüsselnden Datenrahmen diesen Nonce-Wert wiederverwenden.

Phase 4 Für einen erfolgreichen Angriff ist es nun noch erforderlich, den zu korrumpierenden Access-Point ebenfalls davon zu überzeugen, den gleichen *PTK* und *GTK* zu verwenden, den der Opfer-Client in Phase 3 schon installiert hat. Das Problem: Der Opfer-Client hat schon den *PTK* installiert und die letzte Nachricht vier ist schon verschlüsselt. Was den Sicherheitsforschern an dieser Stelle weitergeholfen hat, ist eine sehr laxe Formulierung des IEEE-802.11-Standards [8] welcher wörtlich besagt:

> „On the reception of message 4, the Authenticator verifies that the Key replay Counter Field is one that it used on this 4-way handshake."

Tatsächlich haben die Sicherheitsexperten Vanhoef und Piessens zeigen können, dass eine ganze Reihe von WLAN-Access-Points derart umgesetzt worden sind, dass sie auch unverschlüsselte vierte Nachrichten mit einem alten Replay-Zähler akzeptieren. Damit werden auch auf der Seite des nun ebenfalls korrumpierten Access-Points die gleichen alten *PTK* und *GTK* installiert, die der Opfer-Client schon mit Erhalt der zweiten Nachricht vom Typ 3 verwendet (Abb. 3.14).

3.5.3.5 Verschlüsseltes wiederholtes Senden

Für unseren Angreifer Mallory erweist sich das verschlüsselte wiederholte Senden als deutlich aufwendiger als das unverschlüsselte wiederholte Senden, führt aber schlussendlich ebenfalls zu den erwünschten Effekten. Hierbei sind die folgenden zwei Aspekte besonders hilfreich: Zum einen stellt es sich heraus, dass auf dem WLAN-fähigen Opfer-Client ein wenig genauer betrachtet werden muss welche Komponente zu welchem Zeitpunkt eine Nachricht sendet bzw. erhält. Vanhoef und Piessens zeigen in ihrer Arbeit auf, dass hier zu unterscheiden ist zwischen der (Haupt-)CPU und der Wireless-NIC. Ein wesentlicher Aspekt dabei ist, dass auf der Wireless-NIC zwar das jeweilige zum Einsatz kommende Verschlüsselungsprotokoll umgesetzt ist, also WPA oder WPA2, dort aber nicht der zu verwendende symmetrische paarweise Schlüssel *PTK* gespeichert ist. Zum anderen hat sich unter diesen Gegebenheiten das gezielte Herbeiführen einer Wettlaufsituation (engl. *race-condition*) zweier am Opfer-Client eingehender Nachrichten als hilfreich erwiesen. Das verschlüsselte wiederholte Senden erfolgt in zwei Phasen:

Phase 1 Der MitM-Access-Point blockiert die erste vom Access-Point zum Opfer-Client übertragene Nachricht vom Nachrichtentyp 3.

Client		Mallory		Access-Point
PMK				PMK
	←	N_A, sn	←	wähle N_A
wähle N_S erstelle PTK				
	→	N_S, sn, MIC_{PTK}	→	
				prüfe MIC_{PTK} erstelle PTK
	←	N_A, $sn + 1$, MIC_{PTK}	←	$timer = max$
prüfe MIC_{PTK} installiere PTK				
sende $sn + 1$, MIC_{PTK}	→	!		
	←	N_A, $sn + 1$, MIC_{PTK}	←	wenn $timer == 0$
blockiere $sn + 1$, MIC_{PTK} Re-installiere PTK sn zurücksetzen $nonce$ zurücksetzen				
	→	$sn + 1$, MIC_{PTK}	→	prüfe MIC_{PTK}, installiere PTK

Abb. 3.14 Abfolge des KRACK-Angriffs durch Abfangen der vierten Nachricht und clientseitiges unverschlüsseltes Senden der vierten Nachricht; Darstellung hier nur für ***PTK*** (und nicht auch für ***GTK***). (Quelle: eigene Darstellung in Anlehnung an [16])

Phase 2 In dem Moment, in dem der Access-Point die Nachricht vom Typ 3 das zweite Mal versendet, sendet der MitM-Access-Point ebenfalls die erste zurückgehaltene Nachricht vom Typ 3.

Was bewirkt dieses Vorgehen nun beim Opfer-Client? Die Wireless-NIC des Opfer-Clients leitet beide Nachrichten vom Typ 3 des Vier-Wege-Handshakes an die Warteschlange der Haupt-CPU. Mit Eingang und Bearbeitung der ersten Nachricht vom Typ 3 antwortet die Haupt-CPU und weist die Wireless-NIC an, den Schlüssel ***PTK*** zu installieren. Des Weiteren verfährt die Haupt-CPU wie folgt: Sie nimmt die unverschlüsselte zweite Nachricht vom Typ 3 aus der Warteschlange entgegen. Hierzu ist zu bemerken, dass sowohl Android als auch Linux die Verarbeitung von unverschlüsselten Rahmen des Nachrichtenformates *Extensible Authentication Protocol over Local Area Network* (EAPOL) unterstützen. Da die Wireless-NIC den ***PTK*** jedoch schon aufgrund der vorangegangenen Anweisung der Haupt-CPU installiert hat, erfolgt die anschließende Antwort der Wireless-NIC an den zu korrumpierenden Access-Point nun verschlüsselt (Nonce = 1).

Durch die oben geschilderten Angriffe mittels MitM-Access-Point auf den Vier-Wege-Handshake von WLAN durch wiederholtes Senden einer unverschlüsselten, oder im Bedarfsfall verschlüsselten Nachricht ergeben sich die unterschiedlichsten Auswirkungen, je nachdem, ob TKIP, CCMP oder GCMP bei WPA bzw. WPA2 verwendet werden. Diese betreffen ein wiederholtes Einschleusen von Daten, die Möglichkeit, den

Verkehr zu entschlüsseln, sowie in welche Richtung Daten eingeschleust werden können. Eine detaillierte Aufstellung hierüber ist der Arbeit [16] zu entnehmen. Machen Sie sich bitte klar: Hierdurch ist es im Einzelfall möglich, TCP-Verbindungen zu Zielpunkten im Internet zu übernehmen und Daten in diese Verbindungen zu injizieren oder aber den korrumpierten WLAN-Access-Point als Gateway zu nutzen, um Pakete an andere Geräte im Netzwerk zu injizieren. Damit sind die Auswirkungen eines KRACK-Angriffs immens.

3.5.3.6 Aus der Praxis

Abschließend sei angemerkt, dass eine *Proof-of-Concept*-(PoC-)Implementierung für den Fall des unverschlüsselten wiederholten Sendens in Python von Mathy Vanhoef auf Github zur Verfügung gestellt worden ist [18]. Diesen PoC hat Alexander Eger, Bachelor-Student des Studienganges Unternehmens- und IT-Sicherheit (UNITS) der Hochschule Offenburg, verwendet, um den Angriff nachzustellen. Aus Abb. 3.15 wird ersichtlich, dass die *WPA Key Nonce* sowie der *Key IV* tatsächlich beide auf 000000000000000000… zurückgesetzt worden sind. In dem Versuchsaufbau ist das O(pfer)-Gerät ein HTC One M5 mit Android-Version 5.0.2. Der Access-Point ist ein TP-LINK TL-WR940N mit der Firmware-Version 3.16.9 Build 160620 Rel.49453n und Hardware-Version WR940N v4 00000000. Das Gerät des Angreifers ist mit Kali Linux ausgestattet. Nach dieser erfolgreichen Durchführung des KRACK-Angriffs hängt das weitere Vorgehen davon ab, ob die Ende-zu-Ende-Verbindung unverschlüsselt vorliegt oder ebenfalls verschlüsselt ist, beispielsweise mittels https. Für letzteren Fall müssten dann zum Aufbrechen der Verbindung noch weitere Werkzeuge wie sslstrip verwendet werden.

3.5.4 WPA3

Nicht zuletzt auch um die Verwundbarkeit von WPA2 durch den KRACK-Angriff zu beheben, hat die WiFi-Alliance im Juni 2018 eine Erweiterung von WPA2 mit dem

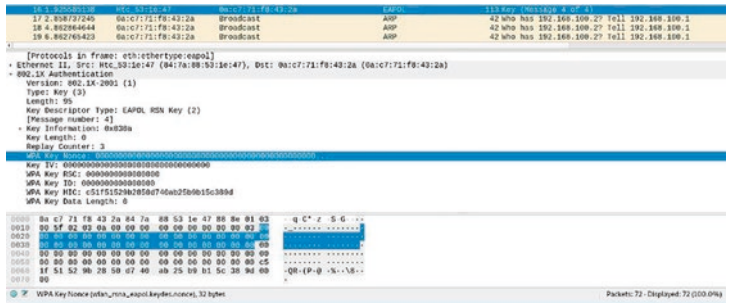

Abb. 3.15 Darstellung des KRACK-Angriffs mittels Wireshark unter Verwendung der PoC-Implementierung von Mathy Vanhoef für den Fall des unverschlüsselten wiederholten Sendens. (Quelle: eigene Darstellung)

Namen WPA3 spezifiziert. Allerdings ist durch die Verabschiedung dieses Standards auch in absehbarer Zeit kein vollständiges Verschwinden aller durch WPA2 gesicherter WLAN-Verbindungen zu erwarten. Vielmehr kann davon ausgegangen werden, dass WPA2 und WPA3 für einen längeren Zeitraum gemeinsam existieren werden. Die ersten WPA3-zertifizierten Geräte sind seit 2019 verfügbar. Dabei unterscheidet der Standard zwischen WPA3-Personal und WPA3-Enterprise, wobei beide Varianten auf WPA2 aufsetzen. Als konkrete Verbesserungen von WPA3 gegenüber WPA2 werden genannt:

Ad1 Die zwingend erforderliche Verwendung eines 192 Bit langen Schlüssels.

Ad2 Durch eine verbesserte Absicherung des Vier-Wege-Handshake-Protokolls zur Schlüsselerneuerung wird dem KRACK-Angriff begegnet.

Ad3 Verbindliche Verwendung von *Protected Management Frames* (PMF) laut 802.11w [10] mit dem Ziel, einige der bisher nicht geschützten Verwaltungsinformationen wie Beacons, Probes, Anfragen zur Authentifizierung und De-Authentifizierung sowie Assoziierung und De-Assoziierung gegen Manipulationen durch Mallory abzusichern.

Derartige Rahmen sind unverschlüsselt, da sie von allen Geräten im Netz nicht nur empfangen, sondern auch gelesen werden sollen. Allerdings erfolgte bisher für solche Rahmen auch keinerlei Authentifizierung, da 802.11w bisher optional und eben nicht verpflichtend einzusetzen war. So ist die gezielte Fälschung von Broadcast-Nachrichten sehr leicht möglich wie beispielsweise durch das in Abschn. 3.5.2.1 beschriebene WLAN-ARP-Spoofing. Um derartigen Angriffen zu begegnen, wird durch den Standard 802.11w hierzu ein neuer Schlüssel, der *Integrity Group Temporal Key (IGTK),* eingeführt, um Broadcast/Multicast-Management-Rahmen zu schützen. Dieser wird nun zusätzlich neben weiteren Schlüsseln innerhalb des Vier-Wege-Handshakes zwischen Client und Access-Point ausgehandelt und kommt anschließend unter Verwendung des *Broadcast Integrity Protocol* (BIP) zum Replay-Schutz und Integritätsschutz von Broadcast-Verwaltungsrahmen zum Einsatz. Die wesentliche Neuerung ist, dass der *IGTK* nun verwendet wird, um einen Message Integrity Code (MIC) zu erzeugen und einzelne Broadcast-Verwaltungsnachrichten mit diesem zu versehen. Allerdings muss man sich auch hier darüber bewusst sein, dass der *IGTK* ein Gruppenschlüssel ist und damit jedem sich am Access-Point korrekt eingewählten Client mitgeteilt wird und diesem somit bekannt ist. Ergo: WLAN-ARP-Spoofing für einen externen Mallory kann mittels PMF tatsächlich unterbunden werden, da dieser nicht im Besitz des *IGTK* ist und somit ARP-Nachrichten nicht unerkannt fälschen kann. Ist Mallory allerdings ein Insider, der selbst im Besitz des *IGTK* ist, indem er sich beim AP angemeldet hat, so funktioniert das in Abb. 3.10 dargestellte WLAN-ARP-Spoofing eines bereits angemeldeten ‚Insider Mallory', um unerkannt Proxy im WLAN-Netzwerk zu werden noch immer. Hierbei gilt $GTS = IGTK$.

Ad4 Eine Authentifizierung mittels *Simultaneous Authentication of Equals* (SAE), um Offline-Angriffe auf verschlüsselte Passwörter mittels Wörterbuchangriffen zu

erschweren, erfolgt auch dann, wenn ein zu kurzes oder anderweitig unsicheres Passwort vom Nutzer verwendet wurde. Verwendet wird hierzu ein sogenannter Zero-Knowledge-Beweis, also ein Protokoll, bei dem die Gegenseite ihr Gegenüber von der Kenntnis eines Geheimnisses überzeugen kann, ohne Informationen über das eigentliche Geheimnis zu offenbaren. Während bei einem Wörterbuchangriff der Angreifer offline agieren kann, ist mittels SAE für einen Angreifer nun eine Online-Interaktion zwingend erforderlich, um an sein Ziel zu gelangen. Das Erfordernis solch einer Online-Interaktion schränkt den Angriffshebel von Wörterbuchangriffen enorm ein.

Ad5 Verwendung von *Opportunistic Wireless Encryption* (OWE) [RFC 8110] von Harkins und Kumari zur Einwahl in Gast-WLANs oder öffentliche Hotspots. Dabei dient OWE als eine bestenfalls moderate Sicherheitsverbesserung zur bisherigen ‚no-authentication' für öffentliche Hotspots unter WPA2.

OWE verwendet während der 802.11-Authentifizierung verschiedene Diffie-Hellman-Varianten zur Schlüsselübereinkunft basierend auf Elliptic Curve Cryptograpy (ECC) oder Finite Field Cryptography (FFC) zur Aushandlung eines symmetrischen Schlüssels. Dieser Schlüssel wird als der Pairwise Master Key (PMK) bezeichnet und dient zur Absicherung des darauf folgenden Vier-Wege-Handshakes. Mit Beendigung des Vier-Wege-Handshakes haben sich Client und Access-Point dann auf die Schlüssel *Key-Encryption Key* (KEK) und *Key-Confirmation Key* (KCK) sowie einen *Message Integrity Code* (MIC) verständigt.

Bezüglich der durch OWE gebotenen Sicherheit drängt sich für mich der Vergleich mit dem Zünden vieler Nebelkerzen auf. So wird dem unbedarften Nutzer meines Erachtens vorgegaukelt, mit OWE eine höhere Sicherheit in öffentlichen Netzen ohne Verwendung eines VPN zu erlangen. Da jede der gebotenen Diffie-Hellman-Varianten bei OWE jedoch ohne eine Authentifizierung von Alice und Bob erfolgt, hat ein aktiver Angreifer Mallory, der sich ebenfalls in der Nähe des öffentlichen Access-Points befindet, jederzeit die Möglichkeit, zum Proxy zwischen Client und Acces-Point zu werden. Die Autoren des RFC's 8110 zu OWE weisen im Abschnitt ‚Security Considerations' auf diesen Sachverhalt ausdrücklich hin. Aber Hand auf's Herz: Wie viele WLAN-Nutzer werden diesen RFC der IETF tatsächlich studiert, geschweige denn verstanden haben, bevor sie sich mit der neuen sichereren WPA3-Variante unter Nutzung von OWE in ein öffentliches WLAN einwählen?

Ad6 Umsetzung des Konzeptes *Perfect Forward Secrecy* (PFS).

Insbesondere WPA3-Enterprise enthält nun ‚auf dem Blatt' zunächst einmal sehr sinnvolle Erweiterungen:

1. **Authentifizierte Verschlüsselung** unter Verwendung des 256-bit Galois/Counter-Mode Protocol (GCMP-256)
2. **Schlüsselableitung und Bestätigung** mittels ECDH und ECDSA unter Verwendung einer 384 Bit elliptischen Kurve.

3. **Robuster Schutz von Management-Rahmen durch** 256-bit Broadcast/Multicast Integrity Protocol Galois Message Authentication Code (BIP-GMAC-256)

Was diese auf den ersten Blick allesamt sinnvollen Sicherheitsverbesserungen bei WPA3-Enterprise dann schlussendlich wert sind, wird uns die Zukunft zeigen. So dürfte durch den robusten Schutz von Management-Rahmen das ARP-Spoofing im WLAN (Abschn. 3.5.2.1) zumindest für einen externen Mallory deutlich schwerer umzusetzen sein. Ein Insider Mallory wird auch unter Verwendung von PMF weiterhin als Proxy agieren können. Inwieweit bei WPA3-Enterprise die Schlüsselableitung mittels ECDH und ECDSA tatsächlich sinnvoll abgesichert ist, hängt insbesondere auch immer davon ab, in welcher konkreten Ausgestaltung ECDH und ECDSA verwendet werden. In Abschn. 4.4.3 werden wir den ‚Angriff über ungültige Kurven' auf Bluetooth LE Secure Connection erörtern. Obwohl auch hier ECDH sowie eine Authentifizierung der Nachrichten erfolgt kann der Angreifer dennoch prinzipiell zu einer sehr hohen Wahrscheinlichkeit auf den mittels ECDH vereinbarten Schlüssel schließen. Es ist also nicht nur die Frage, welche kryptografischen Bausteine und Protokolle verwendet werden, sondern immer auch *wie,* also in welcher konkreten Ausgestaltung, diese zum Einsatz kommen. Da der RFC 8110 für WPA3 jedoch besagt, dass die Kurvenpunkte nach Empfang verifiziert werden müssen, dürfte der in Abschn. 4.4.3 beschriebene Angriff auf ECDH bei Bluetooth nicht so ohne Weiteres auch bei WPA3 anwendbar sein. Allerdings enthält der RFC 8110 nicht mit welchen technischen Mitteln die Punkteverifikation tatsächlich zu erfolgen hat. Schon dies dürfte eine substantielle Schwäche sein. Darüber hinaus werden klassische Seitenkanalangriffe auf ECC und ECDH so wie wir sie in Abschn. 4.4.4 kennen lernen werden, jedoch vermutlich unter WPA3 auch weiterhin noch zum Tragen kommen.

3.6 Zusammenfassung

Lassen Sie uns noch einmal Revue passieren, welche wesentlichen Verwundbarkeiten in drahtlosen lokalen Netzen vorzufinden sind oder aber in der Vergangenheit vorzufinden waren. Begonnen haben wir dieses Kapitel mit grundsätzlichen Bemerkungen zur Funktionsweise der Sicherungsschicht. Danach haben wir mit dem Paket-in-Paket-Injizier-Angriff eine Verwundbarkeit und ihre Auswirkungen kennen gelernt, die immer dann vorhanden ist, wenn die Kommunikation nicht mittels kryptografischer Verfahren gesichert ist oder im Falle von IEEE 802.15.4 eine nicht geeignete AES-Variante zum Einsatz kommt. Im Anschluss an diese Betrachtungen haben wir dann im Rahmen eines kompakten Praxisblocks aufgezeigt, welche Möglichkeiten bestehen, ein proprietäres drahtloses Übertragungsprotokoll zu analysieren, bei dem der Aufbau der Rahmen im Vorfeld nicht über einen Standard oder aber eine anders geartete Spezifikation vorliegt. Im Abschnitt zu Verwundbarkeiten von WLAN sind wir

dann auf die Ursprünge des Sicherheitsprotokolls mit Wired Equivalent Privacy (WEP) und seine Nachfolger WiFi Protected Access (WPA) sowie IEEE 802.11i (WPA2) eingegangen, bevor wir schließlich mit WPA3 die aktuellste und seit 2019 verfügbare Version zur Absicherung von WLAN-Verkehr erörtert haben. Während bei WEP die fehlerhafte Spezifikation des Initialisierungsvektors und dessen teilweise sehr laxe Umsetzung bei der Implementierung sowie die Verwendung eines CRC zur Integritätsprüfung für ein nur sehr lückenhaft abgesichertes Protokoll verantwortlich sind, konnte zumindest sein nicht abwärtskompatibler Nachfolger WPA2 für eine gewisse Zeit als sicher gelten. Dieses lässt sich jedoch nicht für die abwärtskompatible Variante WPA feststellen. Verschlüsselte ARP-Anfragen und -Antworten sowie Informationen über die Länge von Nachrichten und deren Inhalte aufgrund fester Wertebereiche für IP-Adressen ermöglichen einem Angreifer, den Suchraum für den verwendeten WLAN-Schlüssel bei WPA signifikant einzugrenzen. Einzelne Werkzeuge zu einer automatisierten Ausnutzung dieser Verwundbarkeit haben wir im praktischen Teil vorgestellt. Vorab haben wir jedoch auch feststellen müssen, dass sowohl ein mit WPA als auch ein mit WPA2 gesichertes WLAN gegen ARP-Spoofing im WLAN anfällig sind, ein passiver Angreifer also Proxy-Funktionalität im eigenen WLAN erlangen kann, sodass der Verkehr eines weiteren Clients über das Angreifer-Gerät geleitet wird, bevor es an den eigentlichen Access-Point geht. Der noch gar nicht so lange öffentlich bekannte KRACK-Angriff schließlich hat Auswirkungen auf WPA- und auf WPA2-gesicherte WLAN-Verbindungen. Interessant dabei ist: Der Angriff beruht im Kern auf dem Zwei-Armeen-Problem als Dilemma jeglicher Kommunikationsprotokolle. Bisher offenkundig waren jedoch nur die Auswirkungen dieses Dilemmas auf die geöffneten Kommunikationsverbindungen an sich. In Kombination mit der nur wenig formalen Beschreibung der Zustandsübergänge für Client und Access-Point haben wir gesehen, dass hieraus nun aber ein viel größeres Schadenspotenzial erwächst: Ein Angreifer Mallory muss nun nicht mehr das Verschlüsselungsverfahren selbst brechen bzw. mittels Brute-Force-Methode an den Schlüssel gelangen, sondern kann über das Erzwingen eines neuen Re-Keying-Vorganges im besten Fall einen bekannten oder aber schwachen Schlüssel zur nochmaligen Verwendung einspielen. Letztendlich gelingt ihm dies, da nach Ablauf eines Timers eine Nachricht ein weiteres Mal versendet wird, auch auf die Gefahr hin, dass in der Zwischenzeit die erste Nachricht unter Umständen doch noch bei der Gegenstelle eingegangen ist. Zum Abschluss dieses Kapitels haben wir dann den Nachfolger von WPA2 präsentiert. Mit WPA3 hat die WiFi-Alliance im Juni 2018 eine Erweiterung von WPA2 vorgestellt für die wir feststellen konnten, dass dieses Sicherheitsprotokoll vor KRACK-Angriffen schützen kann und gegen WLAN-ARP-Spoofing-Angriffe von externen Angreifern immun ist. Dass mit WPA3 nun das Konzept des Perfect Forward Secrecy umgesetzt wurde, ist ebenfalls als sehr vorteilhaft zu werten. Dennoch bleibt abzuwarten, welche Verwundbarkeiten und daraus resultierenden Angriffe die Sicherheitsexperten mit der Zeit für diese Variante der WLAN-Sicherung entwickeln werden.

Literatur

1. Agrawal, M., Biswas, S., Nandi, S.: Advanced stealth man-in-the-middle attack in WPA2 encrypted Wi-Fi networks. IEEE Commun. Lett. **19**, 581–584 (2015)
2. Biswas, A., Alkhalid, A., Kunz, T., Lung, C.H.: A lightweight defence against the packet in packet attack in ZigBee networks. In: Proceedings of Wireless Days, IFIP, S. 1–3 (2012)
3. Degel, M., Ellsässer, T., Fritscher, D., Iser, A., Kleboth, S.: Packet-in-packet injection – exploiting 802.15.4, Projekt-Dokumentation im Studiengang Computer Networking (2014)
4. Ettercap. http://www.ettercap-project.org/ettercap/. Zugegriffen: 3. Okt. 2018
5. Goodspeed, T.: Phantom boundaries and cross-layer illusions in 802.15.4 digital radio. IEEE Secur. Priv. Workshops, 181–184 (2014)
6. Goodspeed, T.: An Advanced Mitigation Bypass for Packet-in-Packet; Or, I'm Burning 0day to Use the Phrase ‚Eighth of a Nibble'. POC||GTFO and Friends, Philadelphia (2014) (in print)
7. Goodspeed, T., Bratus, S., Melgares, R., Shapiro, R., Speers, R.: Packets in packets: does Orson Welles' in-banding attacks for modern radios. In: Proceedings of the 5th USENIX Conference on Offensive Technologies (2011)
8. IEEE Std 802.11i: Amendment 6: medium access control (MAC) security enhancement (2004)
9. IEEE Std 802.11r: Amendment 2: fast basic service set (BSS) transition (2008)
10. IEEE Std. 802.11w: Protected management frames (2009)
11. Menezes, A.J., van Oorshot, P.C, Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
12. Plumer, D.C.: IETF RFC 826, an ethernet address resolution protocol – or – converting network protocol addresses to 48.bit Ethernet addresses for transmission on ethernet hardware (1982)
13. Sastry, N., Wagner, D.: Security Considerations for IEEE 802.15.4 Networks. ACM WiSe, Philadelphia (2004)
14. Scapy. https://scapy.net/. Zugegriffen: 5. Okt. 2018
15. Tews, E., Beck, M.: Practical Attacks Against WEP und WPA, S. 79–86. ACM WiSec, Philadelphia (2009)
16. Vanhoef, M., Piessens, F.: Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. ACM CCS'17 (2017)
17. Vanhoef, M., Piessens, F.: Advanced Wi-Fi attacks using commodity hardware. ACSAC, 256–265 (2014)
18. Vanhoef, M., Piessens, F.: https://github.com/vanhoefm/krackattacks-poc-zerokey/blob/research/krackattack/krack-all-zero-tk.py. Zugegriffen: 04. März. 2020
19. ZigBee Alliance: *ZigBee specification.* In: ZigBee Document 053474r20 (2012)
20. x00 Sec, Quick n' Dirty ARP Spoofing in Python. https://0x00sec.org/t/quick-n-dirty-arp-spoofing-in-python/487. Zugegriffen: 26. Apr. 2019

4 Verwundbarkeiten in Personal Area Networks

4.1 Bluetooth

Wenn sich nach fast zwei Jahrzehnten der Sicherheitsanalysen auf WLAN verschiedensten WLAN-Generationen noch immer schwerwiegende Verwundbarkeiten aufzeigen lassen, überrascht es den Leser vermutlich nicht, dass Bluetooth mit ähnlichen und teilweise noch drastischeren Angriffsarten zu kämpfen hat. So ist es in diesem Zusammenhang erwähnenswert, dass der WLAN-Standard aus 450 Seiten Spezifikation besteht, während der aktuelle Bluetooth-Standard mittlerweile auf 2882 Seiten Spezifikation angewachsen ist. Sicherlich besteht ein nicht unerheblicher Zusammenhang zwischen der Anzahl der Sonderfälle, Features und Gadgets eines IT-Produktes und dem daraus resultierenden wachsenden Spezifikationsumfang sowie der für die Umsetzung erforderlichen lines-of-code und der Anzahl an Bugs, die sich daraufhin in die Umsetzung einnisten. Diese können das Resultat des Spezifikationswustes oder einer unsauberen Implementierung sein. Solche Sicherheitslücken wiegen schwer, gerade wenn man sich verdeutlicht, dass Bluetooth mehr und mehr im Smart-Home-Bereich verwendet wird, beispielsweise um per Smartphone Haustüren oder Garagentore ‚smart' zu öffnen bzw. die Jalousien an Fenstern zu steuern. Dies ist besonders pikant, da dieselben Smartphones, meistens unbedacht mit aktiviertem Bluetooth, sich in öffentlichen Bereichen wie Straßenbahnen, Zügen oder öffentlichen Plätzen im Übertragungsbereich von Bluetooth-fähigen Geräten von Hackern befinden.

Die erste Version des Bluetooth-Protokolls aus dem Jahre 1999 hat mit Bluetooth 5 aus dem Jahre 2016 so gut wie nichts mehr gemein. Wichtige Zwischenversionen waren im Jahre 2010 v4.0, Bluetooth Low Energy mit einer höheren Reichweite, sowie im Jahre 2014 Bluetooth v4.2 mit LE Secure Connections und Privacy-Verbesserungen. Versionen bis v2.0 aus dem Jahre 2004 waren erst gar nicht gegen Man-in-the-Middle-Angriffe geschützt [12]. Erst mit der Version 2.1 aus dem Jahre 2007 wurde *Secure*

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_4

Simple Pairing (SSP) eingeführt mit dem angestrebten Ziel, einen Schutz gegen Man-in-the-Middle-Angriffe zu bieten. Bluetooth v4.2 nutzt im Protokoll *LE Secure Connections* (SC) eine Variante der in Abschn. 2.4.6 eingeführten Diffie-Hellman Schlüsselübereinkunft. Für LE Secure Connections erfolgt diese jedoch in einer mittlerweile sehr gängigen Variante unter Verwendung von Elliptische-Kurven-Kryptografie. Auf dieses Protokoll zur Schlüsselübereinkunft werden wir später daher noch genauer eingehen. Doch bevor wir uns einzelne Angriffe der jüngsten Vergangenheit näher ansehen, hier noch ein kompakter Überblick über die Pairing-Modi und Pairing-Phasen, so wie sie der Bluetooth-Standard vorsieht.

4.1.1 Pairing-Modi und Pairing-Phasen

Das Verständnis der einzelnen Pairing-Modi und Pairing-Phasen, die Bluetooth unterstützt, ist insbesondere hilfreich, um den später beschriebenen ‚Angriff über ungültige Kurven' ein wenig genauer in die konkrete Abfolge des Bluetooth-Pairings einordnen zu können. Diesen werden wir in Abschn. 4.4.3 ein wenig detaillierter erörtern.

Welcher Pairing-Modus zwischen zwei Geräten verwendet wird, hängt insbesondere davon ab, über welche Fähigkeiten beide Geräte hinsichtlich der Darstellung oder einer Bestätigung durch den Nutzer verfügen. Wird *Numeric Comparison* verwendet, so kommt auf beiden Geräten eine sechsstellige Zahlenfolge zur Anzeige. Durch Anklicken auf beiden Geräten bestätigt der Nutzer die Gleichheit dieser Zahlenfolge auf beiden Geräten.

Dagegen erfordert der Pairing-Modus *Just Works* keinerlei Bestätigung durch Anklicken auf den Geräten. Die geforderte sechsstellige Zahlenfolge besteht in diesem Falle dann lediglich aus lauter Nullen. Damit bietet dieser Pairing-Modus auch keinen Schutz gegen MitM-Angriffe. Unter Verwendung des Modus *Passkey Entry* kommt die Zahlenfolge auf einem Gerät zur Anzeige. Sie muss zur erfolgreichen Durchführung des Pairings dann vom Nutzer auf dem anderen Gerät eingegeben werden. Der Pairing-Modus *Out of Band* hingegen ist nicht zur Gänze im Bluetooth-Standard abgebildet, sodass es den Herstellern obliegt, diesen nach ihrer Interpretation umzusetzen. Die Idee ist, dass Informationen neben der Übertragung via Bluetooth auch noch über einen weiteren Kanal ausgetauscht werden. Dies kann beispielsweise durch das Scannen von QR-Codes oder über *Near Field Communication* (NFC) erreicht werden.

Der eigentliche Pairing-Vorgang sowohl vom Secure Simple Pairing (SSP) als auch von Secure Connections (SC) besteht aus vier Phasen.

Phase 1 **Aushandeln des Pairing-Modus** (engl. *Feature Exchange*)
Die Geräte verständigen sich auf den zu verwendenden Pairing-Modus, indem das schwächere Gerät vorschlägt, welcher der oben genannten Modi zur Anwendung kommt. Dieser Pairing-Modus wird ausgeführt, sodass mit der Beendigung der ersten Phase die beiden Geräte sich auf die gleiche sechsstellige Zahlenfolge verständigt haben.

Phase 2 **Schlüsselaustausch**
Sowohl Secure Connections (SC) als auch Secure Simple Pairing (SSP) verwenden zur Etablierung eines gemeinsamen Schlüssels das Diffie-Hellman-Verfahren zur Aushandlung eines gemeinsamen Schlüssels unter Verwendung von Elliptische-Kurven-Kryptografie (ECDH). Dabei ist die Wahl der konkreten Domänenparameter durch das *National Institute of Standards and Technologies* (NIST) vorgegeben. SC verwendet die Domänenparameter NIST P-256, SSP unterstützt neben den NIST P-256 Domänenparameter ebenfalls NIST P-192 Domain Parameter. Mit der erfolgreichen Ausführung des ECDH-Protokolls haben sich beide Parteien auf einen gemeinsamen symmetrischen Schlüssel *DHKey* verständigt. Wie im Einzelnen die Abfolge von ECDH zur Erlangung des symmetrischen Schlüssels *DHKey* funktioniert und welche konkreten Parameter vorab vereinbart werden müssen, darauf werden wir in Abschn. 4.4.2 zu sprechen kommen. Allerdings sei an dieser Stelle schon angemerkt, dass der reine ECDH in seiner Abfolge keinerlei Überprüfung der Authentizität der übertragenen Nachrichten enthält und damit ausschließlich gegen einen passiven MitM-Angreifer Eve immun ist, nicht aber gegenüber einem aktiven MitM-Angreifer Mallory. Innerhalb seiner Verwendung bei Bluetooth wird die Authentifizierung in der Phase 3 ‚nachgeholt'. Dass dies dennoch zu einer weiteren Klasse von Angriffen geführt hat, werden wir in Abschn. 4.4.3 noch ausführlicher erörtern.

Phase 3 **Authentifizierung**
Die konkrete Abfolge der Phase 3 des Pairing-Vorganges erfolgt in Abhängigkeit davon, welcher der genannten Pairing-Modi tatsächlich verwendet wurde. Wir werden an dieser Stelle nur die Abfolge für die Modi *Numeric Comparison* sowie *Passkey Entry* erörtern, um mit unserer Beschreibung nicht gar zu ausufernd zu werden. Je nachdem welcher dieser Pairing-Modi zum Einsatz kommt, wird eine Reihe von Funktionen $f1(), f2(), \ldots, f6()$ sowie die Funktion $g1()$ oder $g2()$ angewendet. Deren detaillierte Ausgestaltung ist weiter unten aufgeführt.

Verwendung des Pairing-Modus *Numeric Comparison:* Beide Geräte, hier notiert als A(lice) und B(ob), wählen jeweils unabhängig voneinander eine 128 Bit lange Nonce. Wir notieren beide Nonces als N_a, N_b. Dasjenige Gerät, das nicht der Initiator des Pairing-Vorganges ist, berechnet unter Verwendung der Funktion $f4()$ einen Hashwert C_b auf den folgenden Parametern $C_b = f4(PK_ax, PK_bx, N_b, 0)$. Anschließend wird nun dieser Wert C_b an den Initiator des Pairings übertragen. Hierbei notieren PK_ax und PK_bx die jeweiligen x-Koordinaten der beim ECDH in der Phase 2 übermittelten öffentlichen elliptischen Kurvenpunkte. Nun übertragen beide Geräte ihre jeweiligen Nonces N_a, N_b. Mit diesen Informationen ist es dem Initiator-Gerät des Pairing-Vorganges nun möglich, den Hashwert C_b durch eigene Berechnung zu verifizieren. Unter Verwendung der Funktion $g()$ auf beiden Geräten wird nun die sechsstellige Nummer angezeigt. Dabei gehen wiederum beide Nonces sowie die jeweiligen x-Koordinaten der öffentlichen elliptischen Kurvenpunkte ein, also $g(PK_ax, PK_bx, N_a, N_b)$.

Bestätigt der Nutzer diese Nummer, so wird der Pairing-Vorgang von den Geräten als erfolgreich behandelt. Die Abfolge der Authentifizierung in der Pairing-Phase 3 unter Verwendung des Pairing-Modus Numeric Comparison ist in Abb. 4.1 noch einmal dargestellt.

Verwendung des Pairing-Modus *Passkey Entry:* Hier wird der Passkey r_a auf demjenigen der beiden Geräte erstellt, welches die Fähigkeit hierzu besitzt. Anschließend muss dieser Passkey vom Nutzer auf dem zweiten Gerät eingetragen werden. In der Abb. 4.2 und in der weiteren Beschreibung zur Verwendung des Pairing-Modus Passkey Entry wird dieser mit r_b bezeichnet, obwohl dieser Wert im Regelfall mit dem Wert von r_a übereinstimmen sollte. Für jedes einzelne der 20 Bits des Passkeys wird nun wie folgt vorgegangen: Alice berechnet $C_{ai} = f(PK_ax, PK_bx, N_{ai}, r_{ai})$.

und Bob berechnet seinerseits $C_{bi} = f(PK_ax, PK_bx, N_{bi}, r_{bi})$. Nachdem der jeweilige Wert C_{ai} bzw. C_{bi} an die Gegenstelle(n) übertragen wurde übertragen diese das aktuell relevante *i*-te Bit der Nonces N_a bzw. N_b also N_{ai} bzw. N_{bi}. Mit dem Erhalt dieser Informationen ist die jeweilige Gegenstelle nun in der Lage, ihrerseits für die *i*-te Runde den

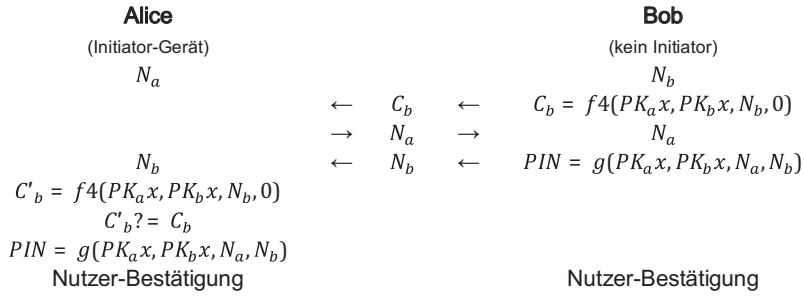

Abb. 4.1 Bluetooth-Authentifizierung unter Verwendung des Pairing-Modus *Numeric Comparison.* (Quelle: eigene Darstellung)

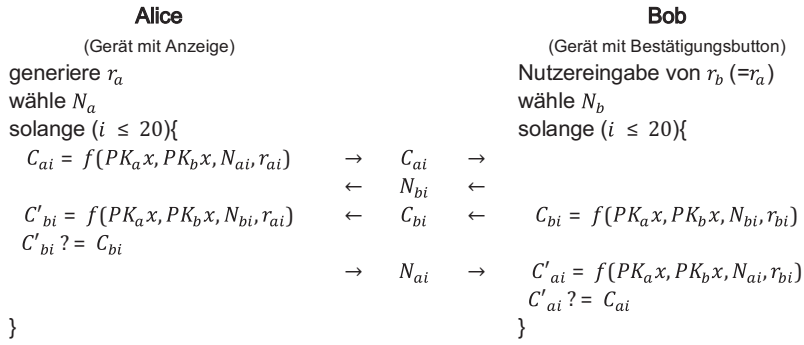

Abb. 4.2 Bluetooth-Authentifizierung unter Verwendung des Pairing-Modus Passkey Entry. (Quelle: eigene Darstellung)

Wert C'_{bi} bzw. C'_{ai} zu berechnen und mit dem empfangenen Wert C_{bi} bzw. den Wert C_{ai} zu vergleichen. Der gesamte Pairing-Vorgang ist nur dann erfolgreich abgeschlossen, wenn auch tatsächlich alle derartigen Prüfungen für jede einzelne Runde fehlerfrei erfolgten.

Phase 4 **Ableitung und Validierung des Sitzungsschlüssels**
In der letzten Phase des Pairing-Vorganges zwischen zwei Bluetooth-fähigen Geräten erfolgen die Ableitung und Überprüfung des Sitzungsschlüssels für die anschließende Verschlüsselung der Daten. Der Sitzungsschlüssel *MACKey* wird unter Verwendung einer Funktion mit Namen *fc*() erstellt. Als einziger geheimer Parameter wird der Schlüssel *DHKey* verwendet. Dieser ist den beiden Geräten als Resultat der Phase 2 bekannt gemacht worden. Hinzu kommen noch als öffentliche Parameter die beiden Nonces N_a und N_b sowie die MAC-Adressen der Geräte, MAC_a und MAC_a: $MACKey = fc(DHKey, N_a, N_b, MAC_a, MAC_b)$. Unter Verwendung von Secure Simple Pairing (SSP) setzt sich die Funktion *fc*() aus den unten beschriebenen Funktionen *f*2() und *f*3() zusammen. Kommt hingegen Secure Connections (SC) zum Einsatz, besteht die Funktion *fc*() aus den Funktionen *f*5() und *f*6(). Die Phase 4 wird mit der Prüfsummenbildung beider Parteien respektive Geräte beendet. Hierzu dient die Funktion *fk*(). Berechnet werden unter Einbeziehung der I/O-Fähigkeiten $IOCap_A$ und $IOCap_B$ beider Geräte die beiden Prüfsummen $E_a = fk(MACKey, N_a, N_b, r_b, IOCap_A, MAC_a, MAC_b)$ und $E_b = fk(MACKey, N_b, N_a, r_a, IOCap_B, MAC_b, MAC_a)$. Für den Fall, dass der Pairing-Modus Passkey verwendet wurde, werden N_a und N_b auf die Bit-Belegungen der letzten Runde gesetzt. Kam hingegen der Pairing-Modus Numeric Comparison zum Einsatz, so sind die Werte für die Parameter r_a und r_b auf null gesetzt.

Im Einzelnen werden in der Phase 2 ‚Schlüsselaustausch' und der Phase 3 ‚Authentifizierung' für die Bluetooth-Pairing-Funktionen die folgenden kryptografischen Bausteine verwendet:

Sofern *Secure Simple Pairing* zum Einsatz kommt:

$$f1(U, V, X, Z) = HMAC - SHA - 256_x(U||V||Z)/2^{128}$$
$$f2(W, N_1, N_2, KeyID, A_1, A_2) = HMAC - SHA - 256_W(N_1||N_2||KeyID||A_1||A_2)/2^{128}$$
$$f3(W, N_1, N_2, R, IOCap, A_1, A_2) = HMAC - SHA - 256_W(N_1||N_2||IOCap||A_1||A_2)/2^{128}$$
$$g(U, V, X, Y) = SHA - 256(U||V||X||Y) mod 2^{32}$$

Sofern *Secure Connections* zum Einsatz kommt:

$$f4(U, V, X, Z) = AES - CMAC_x(U||V||Z)$$
$$T = AES - CMAC_{SALT}(W)$$
$$f5(W, N_1, N_2, A_1, A_2) = AES - CMAC_T(Counter = 0||KeyID||N_1||N_2||A_1||A_2||length = 256)||AES - CMAC_T(Counter = 1||KeyID||N_1||N_2||A_1||A_2||length = 256)||$$
$$f6(W, N_1, N_2, R, IOCap, A_1, A_2) = AES - CMAC_W(N_1||N_2||IOCap||A_1||A_2)$$
$$g2(U, V, X, Y) = AES - CMAC_x(U||V||X||Y) mod 2^{32}$$

Bei den obigen Funktionsbeschreibungen drückt die Operation ‚||' die Konkatenation, also die Zusammenführung von zwei Zeichenfolgen aus. $AES - CMAC_x(M)$ verschlüsselt eine Nachricht M mit dem Schlüssel X unter Verwendung der Blockchiffre AES im Modus *Cipher-based Message Authentication Code* (CMAC). Die Notation $HMAC - SHA - 256_X$ besagt, dass als Message Authentication Code (MAC) das Konstrukt eines HMAC unter Verwendung des symmetrischen Schlüssels X zum Einsatz kommt. Hierbei wird als Hashfunktion der SHA-256 mit einer Hashlänge von 256 Bit aus der SHA-2-Familie verwendet. Weitere Details zur prinzipiellen Wirkungsweise von Hashfunktionen und Message Authentication Codes entnehmen Sie bei Bedarf bitte dem Abschn. 2.4 dieses Buches.

Die in den folgenden Abschnitten beschriebenen Angriffe sind mehrheitlich durch einen sogenannte *Proof of Concept* (PoC) für Bluetooth 5.0 nachgewiesen. Da dieser Standard in seiner aktuellen Version bei Sichtkontakt sogar Reichweiten von bis zu 200 Metern ermöglichen soll und innerhalb eines Gebäudes immerhin noch bis zu 40 m an Reichweite möglich sind, offenbart dies das mögliche Schadenspotenzial. Nicht zuletzt, wenn man noch bedenkt, dass es sich hierbei um einen Markt von 8,2 Mrd. Geräten handelt. Bei den nachfolgend beschriebenen Angriffen ist meist erforderlich, dass der Angreifer sich zumindest zu einem bestimmten Zeitpunkt in der Übertragungsreichweite zwischen zwei Bluetooth-fähigen Geräten befindet. Doch vorab wollen wir noch auf eine Umsetzung in Bluetooth zu sprechen kommen, mit der eine Privacy-freundliche Verwendung von Bluetooth möglich sein soll.

4.2 LE Privacy

Damit ein Bluetooth-fähiges Gerät anderen Geräten in seiner aktuellen Aufenthaltsumgebung mitteilen kann, dass es verfügbar ist, sendet dieses von Zeit zu Zeit Broadcast-Pakete. Solche *Advertisment-Beacons* enthalten die MAC-Adresse des Gerätes. Diese 48 Bit lange MAC-Adresse wird für gewöhnlich vom Hersteller statisch vergeben. Sie kann bis auf Weiteres als weltweit eindeutig angesehen werden und bietet somit die Möglichkeit, das Bewegungsprofil eines Gerätes und damit mit hoher Wahrscheinlichkeit auch das seines Besitzers nachzuverfolgen. Die Umsetzung einer derartigen Angriffsform ist für einen Angreifer Eve sehr einfach durchführbar, da Broadcast-Nachrichten bei Bluetooth typischerweise nicht verschlüsselt übertragen werden und die MAC-Adresse damit als Klartext in jedem zu übertragenden *Advertisement Beacon* vorliegt. Mit Bluetooth Low Energy enthält der Standard nun allerdings die Ausarbeitung einer Gegenmaßnahme, die sich *LE Privacy* nennt und das Ziel verfolgt die Bekanntmachung eines Gerätes in seiner aktuellen Umgebung zu unterstützen, ohne dass damit zwangsläufig das Nachverfolgen der Position eines Gerätes einhergehen muss. Dem Nutzer werden vier unterschiedliche Adresstypen zur Auswahl angeboten. Diese an den Nutzer delegierte Auswahl des Adresstyps ist dem Spannungsfeld geschuldet, das sich oftmals aus der gleichzeitigen Umsetzung von Funktionalitätsaspekten auf der einen Seite und Datenschutz bzw. Privatheit auf der anderen

Seite ergibt. Denn würde ein Bluetooth-fähiges Gerät bei jedem einzelnen *Advertisement*-Vorgang eine jeweils neue, zufällig gewählte 48 Bit-Adresse vergeben und mitteilen, dann wäre dies im Sinne der Privatheit und des Datenschutzes vorbildlich, doch gleichzeitig auch äußerst unkomfortabel und somit als benutzungsunfreundlich zu bewerten. Denn alle vormals zwischen zwei Geräten erfolgreich durchgeführten Pairing-Vorgänge wären wertlos und müssten demnach immer wieder von vorne durchgeführt werden. Nach diesen einleitenden Bemerkungen sind die im Standard vorgesehenen Adresstypen für den Leser nun wohl besser nachvollziehbar: Neben der ursprünglichen MAC-Adresse, bestehend aus dem *Lower Address Part* (LAP), dem *Upper Address Part* (UAP) und einem *Nonsignificant Address Part* (NAP), die fest vom Hersteller vergeben wird und eben keine Privatheit hinsichtlich des Bewegungsprofils eines Gerätes und seines Nutzers bietet, kann der Nutzer wählen zwischen einem zufälligen, aber statischen Adresstyp, einem zufälligen Adresstyp, der nicht mehr auflösbar ist, sowie einem zufälligen Adresstyp, der jedoch für einzelne Geräte auflösbar ist. Letztere Variante erfolgt durch die Verwendung eines Schlüssels, den sogenannten *Identity Resolution Keys* (IRK). Eine zufällig gewählte Adresse wird unter Nutzung des IRK verschlüsselt, sodass all diejenigen Geräte, die im Besitz dieses Schlüssels sind, die Identität des *Beacon* sendenden Gerätes ermitteln können, während alle übrigen Geräte, die nicht diesen Schlüssel besitzen, dessen Identität nicht so ohne Weiteres in Erfahrung bringen können. Mark Loveless hat in seinem Blog [16] zu Bluetooth Security einige der oben genannten Aspekte zu LE Privacy zusammengetragen und ein wenig genauer analysiert. Allerdings weisen die Sicherheitsexperten Scott Lester und Paul Stone [15] darauf hin, dass die Verwendung eines zufälligen Adresstyps, sei er nun auflösbar oder nicht, eine sehr trügerische Form der Privatheit des Bewegungsprofils eines Gerätes bzw. seines Eigentümers bietet. Denn auch wenn die Adresse bei jedem *Advertisement-Beacon* geändert werden würde, so enthalten solch ein *Advertisement-Beacon* sowie, später innerhalb der Kommunikation nachfolgende Beacons eine Reihe weiterer Datenfelder, die statisch sind und aufgrund ihrer Verwendung einem Angreifer Eve genügend Anhaltspunkte bieten dürften, das Gerät zu bestimmen und schlussendlich auf die Identität des Nutzers zu schließen. Beispiele hierfür sind der *Channel Access Code* (CAC) sowie der *Device Access Code* (DAC), die, auch wenn sie nicht zwingend eindeutig sind, Eve doch genügend Informationen zur Erstellung von Bewegungsprofilen liefern. Weitere Beispiele hierzu finden sich in [15]. Alles in allem dürfte damit LE Privacy dem Nutzer eher Privatheit vorgaukeln, als dass es diese tatsächlich gewährleisten könnte.

▶ Die Tatsache, dass auch unter Verwendung von *LE Privacy* mit zufällig gewählten Adressen sowie deren häufiger Abänderung dennoch keine Verschleierung der Identität sowie des Bewegungsprofils eines Bluetooth-Nutzers gegeben ist, liegt daran, dass Werte wie der *Channel Access Code* sowie der *Device Access Code* einem Angreifer Eve noch genügend Informationen bereitstellen, um Rückschlüsse auf den Nutzer und sein Bewegungsprofil zu geben. Diese Verwundbarkeit resultiert aus einer lückenhaften Ausgestaltung von LE Privacy im Bluetooth-LE-**Standard.**

4.3 Blueborne

Unter dem Kunstwort Blueborne gebildet aus den Begriffen *blue* and *airborne,* wird eine Reihe von Verwundbarkeiten zusammengefasst. Im Jahre 2017 wurden diese in einige *Common Vulnerabilities and Exposures* (CVE) der MITRE [19] aufgenommen.

1. Information Leak Vulnerability (CVE-2017-0785)
2. Remote Code Execution #1 (CVE-2017–0781)
3. Remote Code Execution #2 (CVE-2017-0782)
4. Man in the Middle attack #1 (CVE-2017-0783)
5. Man in the Middle attack #2 (CVE-2017-8628)
6. Information leak vulnerability (CVE-2017-1000250)
7. Stack overflow in BlueZ (CVE-2017-1000251)
8. Remote Code Execution via Apple's Low Energy Protocol (CVE-2017-14315)

Jedes Gerät bei dem Bluetooth aktiviert ist, ist anfällig gegen derartige Angriffe. Dies umfasst zum Zeitpunkt der Bekanntmachung der einzelnen CVEs durch das Unternehmen Armis [23] ca. 8,2 Mrd. Geräte. Die Autoren stufen die Verwundbarkeiten in sehr ernst sowie begründbar ernst ein, wobei die in CVE-2017-0781, CVE-2017-0782, CVE-2017-1000251 und CVE-2017-14315 beschriebenen Verwundbarkeiten als sehr ernst und die Verwundbarkeiten der CVE-2017-0785, CVE-2017-0783, CVE-2017-8628 sowie CVE-2017-1000250 von den Sicherheitsexperten als begründbar ernst eingeschätzt worden sind. Nachfolgend wollen wir exemplarisch zwei dieser acht CVEs auf Bluetooth erläutern.

4.3.1 Entfernte Codeausführung

Der unten angeführte Programmcode stammt aus [23], ist als Remote Code Execution#1 eingeordnet und dokumentiert den Einsatz der C++-Funktion `memcpy()`. Durch den Aufruf dieser Funktion können eine Reihe Bytes aus einer Quelle direkt in den Speicherbereich eines Zieles kopiert werden. Dies ist in der Abb. 4.3 dargestellt,

Es lässt sich vermuten, dass der Funktionsaufruf von `memcpy()` mit obigen Parametern im ungünstigen Fall zu einem Pufferüberlauf führen kann. Um diese Implementierungsschwäche ausnutzen zu können, ist es allerdings erforderlich, dass diese Instruktionsfolge auch tatsächlich zur Ausführung gelangt. Die Sicherheitsforscher Ben Seri und Gregory Vishnepolsky haben herausgefunden, dass dies immer dann der Fall ist, wenn man eine speziell geformte Nachricht über eine bestehende Bluetooth-Verbindung sendet.

Immer dann, wenn Kontrollnachrichten des Formates *Bluetooth Network Encapsulation Protocol* (BNEP) [9] eingehen, und zwar dann, wenn zwei BNEP-Kontrollnachrich-

```
UINT *p = (UINT8 *)(p_buf + 1) + p_puf->offset;
…
type = *p++;
extension_present = type >> 7;
type &= 0x7f;
…
switch (type)
{
…
case BNEP_FRAME_CONTROL:
        ctrl_type = *p;
        p = bnep_process_control_packet (p_bcb, p, &rem_len, FALSE);
        if (ctrl_type == BNEP_SETUP_CONNECTION_REQUEST_MSG &&
            p_bcb->con_state != BNEP_STATE_CONNECTED &&
            extension_present && p && rem_len)
      {
        P_bcp->p_pending_data = (BT_HDR *)osi malloc(rem_len);
        memcpy((UINT8 *)(p_bcb->p_pending_data + 1), p, rem_len);
        …
      }
```

Abb. 4.3 Verwundbarkeit durch fehlerhafte Verwendung von `memcpy()`. (Quelle: eigene Darstellung nach [23])

ten in genau einer L2CAP-Nachricht enthalten sind, gelangt die obige Programmsequenz zur Ausführung. Dabei steht das Kürzel L2CAP für das sogenannte *Logical Link Control and Adaptation Protocol.* Aufgabe dieses Protokolls ist es, bei der Umformung roher Eingangsdaten in BLE-Rahmen die höheren Schichten des Bluetooth Protokoll-Stacks mit den unteren Schichten zu harmonisieren. Der Pufferüberlauf kann nun immer dann ausgelöst werden, wenn man Mallory die in Abb. 4.4 aufgeführte Nachricht über eine BNEP-Verbindung einschleust. Dies führt dazu, dass auf einem derart korrumpierten Gerät das entfernte Aufrufen von Programmen möglich wird.

Mittels einer PoC-Implementierung haben nun die beiden Sicherheitsforscher gezeigt, wie beispielsweise die Kamera des Opfergerätes aktiviert werden kann und per Initiierung aus der Ferne Bilder aufgenommen werden können, oder das Mikrofon aktiviert werden kann, und dergleichen mehr.

Type	**Ctr_type**	**Len**	**Overflow Payload** (8 bytes)							
81	01	00	41	41	41	41	41	41	41	41

Abb. 4.4 BNEP-Paket, das den Pufferüberlauf beim Empfänger auslöst. (Quelle: eigene Darstellung nach [17])

Eine derartige Implementierung wirft schon die Frage auf, wie es passieren kann, dass all die Jahre bei keinem einzigen Test und keiner Qualitätsprüfung ein derartiger Fehler aufgefallen ist, zumal die Problematik um Funktionsaufrufe wie `memcpy()` und ähnliche Funktionen bzgl. eines drohenden Pufferüberlaufs schon seit geraumer Zeit bekannt ist und darüber hinaus eine Verwendung derartiger Aufrufe mit Verfahren der statischen Codeanalyse leicht zu beheben gewesen wäre. Kritikwürdig ist also nicht so sehr die Tatsache, dass in Tests die Ausführung dieses Programmzweiges offensichtlich nicht zur Ausführung gelangte, sondern schlicht das Vorhandensein des als sicherheitskritisch zu bewertenden Funktionsaufrufes an sich.

Abschließend noch einige allgemeine Bemerkungen zu potenziellen Verwundbarkeiten und Sicherheitslücken, die sich aus der Verfälschung des Arbeitsspeichers durch die Programmierung mit speicherunsicheren Programmiersprachen wie C und C++ ergeben können: Hier wird der Zugriff auf den Speicher nicht geprüft, sodass der Entwickler bei der Implementierung eines Programms die Low-Level-Speicherverwaltung berücksichtigen sollte. So können unter Umständen, wie bei obiger Blueborne-Verwundbarkeit geschehen, Pufferüberläufe bei fester Pufferlänge schwerwiegende Auswirkungen haben und neben einem Absturz oftmals auch zur Ausführung von Schadcode genutzt werden. Letzteres immer dann, wenn der überschriebene Wert eine Rücksprungadresse aus einer Funktion enthält.

Ein Pufferüberlauf geschieht immer dann, wenn im Programm keine oder aber eine fehlerhafte Prüfung auf einen Puffer fester Länge erfolgt, da zum Zeitpunkt der Allokation des Speicherplatzes für die Daten deren genaue Länge noch gar nicht bekannt ist. Im Falle der Blueborne-Verwundbarkeit Remote Code Execution#1 Verwundbarkeit wird dem Heap-Speicher zunächst ein Speicherplatz der Größe rem_len allokiert. Nun wird jedoch die Funktion `memcpy()` `auf` `p_pending_data` `+1` mit der Größe `rem_len` angewandt was zu einem Pufferüberlauf durch `p_pending_data` führen kann. Diesen Implementierungsfehler haben die Sicherheitsforscher herausgefunden und durch Senden der BNEP-Nachricht an das Opfergerät provoziert und ausgenutzt.

Die Abb. 4.5 skizziert diesen Vorgang einer Speicherverfälschung durch Pufferüberlauf bei fester Pufferlänge noch einmal, so wie diese im Falle der Blueborne-Verwundbarkeit CVE-2017-0781 ausgenutzt worden ist.

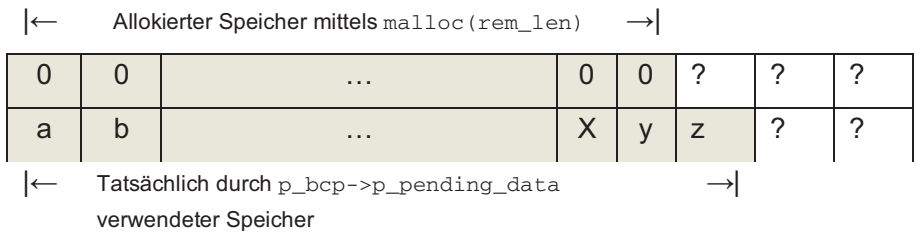

Abb. 4.5 Allokierter Speicher und tatsächlich benötigter Speicher. (Quelle: eigene Darstellung in Anlehnung an [23])

- Die Blueborne-Verwundbarkeit Remote Code Execution#1 Verwundbarkeit beruht auf einer fehlerhaften Verwendung der Funktion `memcpy()` und einem sich hieraus ergebenden Pufferüberlauf im Falle zweier eingehender speziell geformter BNEP-Kontrollnachrichten in genau einer L2CAP-Nachricht. Die Verwundbarkeit resultiert aus einer **fehlerhaften Implementierung** durch überholte Funktionsaufrufe sowie einer fehlerhaften Längenprüfung.

4.3.2 Ein einfacher Man-in-the-Middle-Angriff

Diese unter dem Namen „Bluetooth Pineapple“ bekannt gewordene Schwachstelle nutzt den ‚Just Works‘ Authentifizierungsmechanismus für einen MitM-Angriff aus, der vornehmlich im IoT-Bereich zum Tragen kommen dürfte, also immer genau dann, wenn diverse eingebettete Geräte mit Bluetooth-LE-Funktionalität über einen Pairing-Mechanismus angebunden werden sollen. Diese Form der Authentifizierung erfolgt immer dann, wenn zumindest eines der beiden Bluetooth-fähigen Geräte über keinerlei Display verfügt, auf dem ein PIN angezeigt werden könnte, so wie dies für die anderen (siehe Abschn. 4.1.1) zu bevorzugenden Bluetooth-Authentifizierungsmechanismen erforderlich wäre. Neben *‚Just Works‘* sind als weitere Möglichkeiten der Assoziierung im Standard genannt: *Numeric Comparison* (nur für LE Secure Connections), *Passkey Entry* sowie *Out of Band* (OOB), wobei bei Letzterem ein weiterer herstellerspezifischer vertrauenswürdiger Kanal zur Etablierung eines gemeinsamen Sitzungsschlüssels zu verwenden ist.

Doch bleiben wir bei der ‚Just Works‘-Variante zur Anbindung von Bluetooth-fähigen IoT-Geräten ohne Möglichkeit einer Anzeige von Informationen auf einem Display. Hierzu steht in der Bluetooth-Spezifikation der Version 5.0 oder bei Bluetooth Low Energy (BLE) von 2016:

> „The Just Works association model uses the Numeric Comparison protocol but the user is never shown a number and the application may simply ask the user to accept the connection (exact implementation is up to the end product manufacturer)…The Just Works association model provides the same protection as the Numeric Comparison association model against passive eavesdropping but offers no protection against the MITM attack…“

Wie genau die Umsetzung der Authentifizierung tatsächlich zu erfolgen hat, wird also bei einer ‚Just Works‘-Authentifizierung weg von der Spezifikation hin zu den Herstellern von Bluetooth-Produkten verlagert. Mehr noch: Die Spezifikation besagt unumwunden, dass MitM-Angriffe eines aktiven Angreifers Mallory von den Herstellern erst gar nicht zu unterbinden sind, um ein Bluetooth-5.0-konformes Produkt zur Marktreife zu bringen.

Dies hat im Falle von Android zu der folgenden Implementierung geführt [23]:

Der Code in Abb. 4.6 bedeutet, dass unter Android ein Nutzer ‚Just Works‘ automatisch akzeptiert, immer genau dann, wenn die folgenden Bedingungen erfüllt sind:

```
/* If JustWorks auto-accept */
If (p_ssp_cfm_req->just_works) {
  // Piring consent for JustWorks needed if:
  // 1. Incoming (non-temporary) pairing is detected AND
  // 2. Local IO capabilities are DisplayYeSNo AND
 // 3. remote IO capabilities are DisplayOnly or NoInputNoOutput;
 If (is_incoming && pairing_cb.bond_type != BOND_TYPE_TEMPORARY &&)
     ((p_ssp_cfm_req->loc_io_caps == HCI_IO_CAP_DISPLAY_YESNO) &&
      (p_ssp_cfm_req->rmt_io_caps == HCI_IO_CAP_DISPLAY_ONLY ||
       p_ssp_cfm_req->rmt_io_caps == HCI_IO_CAP_NO_IO))) {
   BTIF_TRACE_EVENT(
       "%s: User consent needed for incoming pairing request. loc_io_caps: "
       "%d, rmt_io_caps: %d",
       __func__, p_ssp_cfm_req->loc_io_caps, p_ssp_cfm_req->rmt_io_caps);
 } else {
   BTIF_TRACE_EVENT("%s: Auto-accept JustWorks pairing", __func__);
   btif_dm_ssp_reply(&bd_addr, BT_SSP_VARIANT_CONSENT, true, 0);
   return;
 }
}
```

Abb. 4.6 Just-Works-Implementierung unter Android. (Quelle: eigene Darstellung in Anlehnung an [23])

i) das Pairing ist temporär, ii) das Android-Gerät muss `DisplayYesNo` als Fähigkeit haben und iii) das Gerät, auf das sich verbunden werden soll, hat die Fähigkeit `Display Only` oder `no IO`.

Damit ist ein Angreifer, der sich zum Zeitpunkt des Pairings in der Übertragungsreichweite dieser beiden Geräte aufhält, nun in der Lage, ein Pairing mit einem Opfergerät ohne irgendeine Interaktion seitens des Nutzers zu erzwingen. Das einzige, was der Angreifer leisten muss, ist, auf das Bluetooth-Kommando Mensch-Computer-Interaktion (HCI) ‚`IO Capability Request`' mit „`NoInputNoOutput, MitM Protection Not Required – No Bonding`" zu antworten. Nachdem auf diese Weise der ‚Just Works'-Pairing-Mechanismus erzwungen wurde kann der Angreifer von nun an auf ausgesuchte Dienste des Opfergerätes zugreifen.

▶ Die Blueborne-Verwundbarkeit Man in the Middle Attack#1 (Bluetooth Pineapple) beruht auf der Tatsache, dass der ‚Just Works'-Mechanismus ohne Authentifizierung der Gegenseite ausgelöst wird. Die Verwundbarkeit gegenüber einem aktiven MitM-Angreifer wurde billigend in Kauf genommen und in der Spezifikation wurde sogar darauf hingewiesen. Die Verwundbarkeit resultiert aus einer **bewussten Verlagerung der Verantwortung weg von der Spezifikation hin zu den Herstellern,** bei denen die Möglichkeit der

> Anbindung unterschiedlicher leistungsschwacher Geräteklassen an ein Bluetooth-Netzwerk offenbar einen höheren Stellenwert erhalten hat, als die Gewährleistung eines begründbaren Sicherheitsniveaus.

Der Pairing-Modus ‚Numeric Comparison' hingegen nutzt zum Aushandeln von symmetrischen Schlüsseln das vielfach bewährte asymmetrische Schlüsselübereinkunft-Protokoll *Elliptic Curve Diffie-Hellman* (ECDH). Dieser Pairing Modus ist damit die deutlich zu bevorzugende Variante gegenüber ‚Just Works', sofern dieser Modus tatsächlich von beiden Geräten unterstützt wird. Allerdings hat es sich gezeigt, dass auch dieser Pairing-Modus hinsichtlich der gebotenen Sicherheit nicht ganz unproblematisch ist. Die Erörterung dieser Verwundbarkeiten zu soll Gegenstand in den nun folgenden Abschnitten sein.

4.4 Ungültige Kurvenpunkte und geänderte Punktkoordinaten

4.4.1 Einleitende Bemerkungen

Eine CVE vom Juli 2018 besagt, dass die Elliptische-Kurven-Parameter nicht immer gründlich genug geprüft werden. Dieser Verwundbarkeit wurde die Kennung CVE-2018-5383 zugewiesen. Mittlerweile ist ein Update der Spezifikation verfügbar, mit dem diese Schwäche behoben wurde. Dies soll uns jedoch nicht daran hindern, auf die ursprünglich vorhandene Verwundbarkeit ein wenig genauer einzugehen. Denn nur so kann man lernen und nachvollziehen, über welch vielfältige Stellschrauben gewiefte Angreifer verfügen, um die Sicherheit eines Systems zu korrumpieren. Die hier erörterte Verwundbarkeit kann dazu führen, dass bei einigen Plattformen es einem Angreifer in Übertragungsreichweite – selbst bei Verwendung von *LE Secure Connections* (LE SC) bzw. *Secure Simple Pairing* (SSP) – gelingen kann, die Verbindung zu übernehmen, indem er auf den ausgehandelten Sitzungsschlüssel schließen kann. Dies gilt sowohl bei einer Authentifizierung mittels ‚Numeric Comparison' als auch mittels ‚Passkey Entry'. Erstere Variante wird gewählt, wenn beide Geräte eine sechsstellige Dezimalzahl anzeigen können und mindestens eines der involvierten Bluetooth-fähigen Geräte diese bestätigen oder verwerfen kann. Die zweite Variante ist genau dann vorzuziehen, wenn zumindest eines der Geräte eine Eingabe durch den Nutzer erhalten kann, während das andere Gerät in der Lage ist, die Eingabe anzuzeigen.

Unter diesen Gegebenheiten wäre es einem Angreifer möglich, die Kommunikation zu entschlüsseln sowie eigene Daten unerkannt in die Verbindung einzuschleusen. Auf diese Verwundbarkeit, die von den israelischen Sicherheitsforschern Eli Biham und Lior Neumann aufgezeigt wurde, hat die Bluetooth Special Interest Group (SIG) umgehend reagiert, indem sie in der Spezifikation die Validierung öffentlicher Schlüssel als zwingend erforderlich angepasst hat. Sie betrifft die MitM-Authentifizierung mittels ECDH

von Qualcomm, Broadcom, Intel sowie Google und basiert auf einem Angriff aus der Kategorie ‚ungültige Kurve'. Mehr Details zu dieser Kategorie von Verwundbarkeiten werden verständlicher nachdem wir einige grundsätzlichen Bemerkungen zur Diffie-Hellman-Schlüsselübereinkunft auf der Grundlage elliptischer Kurven vorgenommen haben.

4.4.2 Varianten der Diffie-Hellman-Schlüsselübereinkunft

4.4.2.1 Diffie-Hellman-Schlüsselübereinkunft mit elliptischen Kurven

Aufgrund ihrer deutlich kleineren erforderlichen Schlüssellängen bei einem vergleichbaren Sicherheitsniveau hat sich die in den 80er-Jahren von Neal Koblitz und Victor Miller vorgeschlagene Elliptische-Kurven-Variante der Diffie-Hellman-Schlüsselübereinkunft ECDH in den letzten Jahren etabliert. ECDH steht dabei für *Elliptic Curve Diffie-Hellman.* Eine Schlüsselübereinkunft mittels ECDH kommt bei Bluetooth im Modus LE Secure Connections zum Einsatz. Eine elliptische Kurve E über einem endlichen Körper K_q der Ordnung q kann notiert werden als $y^2 = x^3 + ax + b$ mit der Einschränkung, dass $2^2a^3 + 3^3b^2 = 0$ gilt. Sie besteht aus der Menge aller Paare (x, y) die Lösung dieser Gleichung sind. Die Kurve E kann über den reellen Zahlen, also $x, y \in \mathbb{R}$ oder über einem endlichen Körper definiert werden. Letztere Darstellung ist für kryptografische Belange von Bedeutung und soll uns daher hier vornehmlich interessieren. Hier wird die Beschreibung der Kurve E um den Modulo-Operator ergänzt, sodass $E : y^2 = x^3 + ax + b \ mod \ q$. Um nun E für kryptografische Zwecke verwenden zu können bedarf es noch eines Generatorpunktes G. Dieser hat die Eigenschaft alle Elemente einer zyklischen Gruppe auf der Kurve zu erzeugen. Das neutrale Element $\mathfrak{o}$ der Gruppe ist dann ein abstrakter unendlich weit entfernter Punkt. Dieser kann entweder ‚plus' unendlich gen y-Achse auf dem Koordinatensystem, oder in ‚minus' unendlich gen y-Achse auf dem Koordinatensystem positioniert sein. Für das inverse Element $-P$ eines Punktes $P = (x, y)$ auf der Kurve gilt dann $P + (-P) = \mathfrak{o}$, wobei über dem Primkörper $GF(p)$ gilt, dass $-P = (x, p - y)$.

Zur Verwendung in Bluetooth wird die elliptische Kurve E gemäß dem Standard FIPS 186-2 [21] des *National Institute of Standards and Technologies* (NIST) spezifiziert. Beide Parteien haben sich dabei vor der Ausführung des eigentlichen Protokolls über die Parameter p, a, b, G und n zu verständigen. Diese Parameter werden in den nachfolgenden Beschreibungen als Domänenparameter, kurz *dom,* zusammengefasst.

Die prinzipielle Sicherheit des Verfahrens beruht hierbei auf dem Problem des diskreten Logarithmus in elliptischen (ECDLP), das besagt, dass es schwer, ist eine ganze

positive Zahl d aus $1 \leq d \leq \#E$ zu finden, sodass ein Punkt G auf einer elliptischen Kurve d-mal mit sich selber addiert einen weiteren Punkt T auf der elliptischen Kurve liefert:

$$G + G + \ldots + G = dG = T$$

Der Parameter $\#E$ notiert die Anzahl der Punkte auf der elliptischen Kurve über dem endlichen Körper K_p und ist laut *Hasses Theorem* auf das Intervall $p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$ begrenzt [22].

Der Ergebnispunkt $C = (x_C, y_C) = A + B$ zweier Punkte $A = (x_A, y_A)$ und $B = (x_B, y_B)$ auf der Kurve E ist auf den ersten Blick ein wenig ‚willkürlich' über seine Koordinaten zu berechnen:

$$x_C = s^2 - x_A - x_B \; mod \; p$$
$$y_C = s(x_A - x_C) - y_A \; mod \; p$$

Die Steigung s ist dabei

$$s = \frac{y_B - y_A}{x_B - x_A} mod \; p$$

für $A \neq B$ und

$$s = \frac{3x_A^2 + a}{2y_A} mod \; p$$

wenn die zu addierenden Punkte identisch sind [22].

An dieser Stelle sei angemerkt, dass die grafische Darstellung der Addition zweier Punkte deutlich leichter nachvollziehbar ist. Sie kann beispielsweise durch die Verwendung des Werkzeugs CrypTool [7] dargestellt und veranschaulicht werden. Durch das Ziehen einer Geraden durch die Punkte A und B ergibt sich ein dritter Schnittpunkt der Geraden mit der Kurve E. Dieser Schnittpunkt gespiegelt an der x-Achse ergibt den Punkt $C = (x_C, y_C)$. In ähnlicher Weise ist bei der Punktdopplung, also der Addition zweier identischer Punkte, vorzugehen.

Die mit CrypTool erstellten Abb. 4.7 und 4.8 veranschaulichen dies für die beispielhafte Kurve $y^2 = x^3 - 5x + 15$, und die Punkte P mit den Koordinaten (−3,0, 1,73) und Q mit den Koordinaten (0,61, 3,48). Der Ergebnispunkt R hat die Koordinaten (2,62, −4,45), wenn eine Punktaddition durchgeführt wurde. Die Punktdopplung des Punktes P mit den Koordinaten (−1,65, 4,32) mit sich selbst, also $R = 2P$, ergibt dann den Punkt R mit den Koordinaten (3,44, −6,21). Diese Darstellung ergibt sich für x und y über den reellen Zahlen, also $x, y \in \mathbb{R}$.

Das eingangs erläuterte ECDLP findet seine Anwendung für kryptographische Verfahren indem man d als privaten Schlüssel, und T als öffentlichen Schlüssel versteht. Der Punkt G ist ein öffentlich bekannter Generatorpunkt, mit dem jeder einzelne Punkt des endlichen Körpers durch Addition mit sich selbst zu erlangen ist.

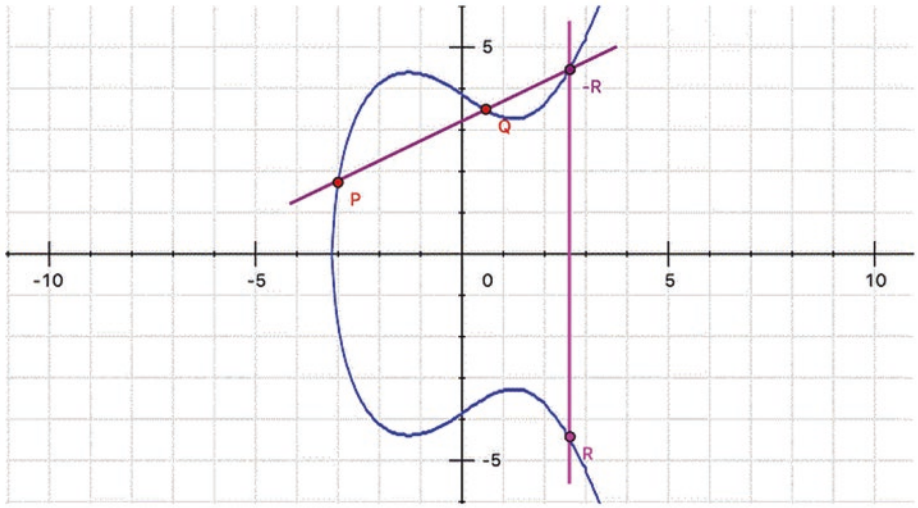

Abb. 4.7 Grafische Darstellung einer Punktaddition der Punkte P und Q zu $R = P + Q$. (Quelle: eigene Darstellung unter Verwendung von Cryptool [7])

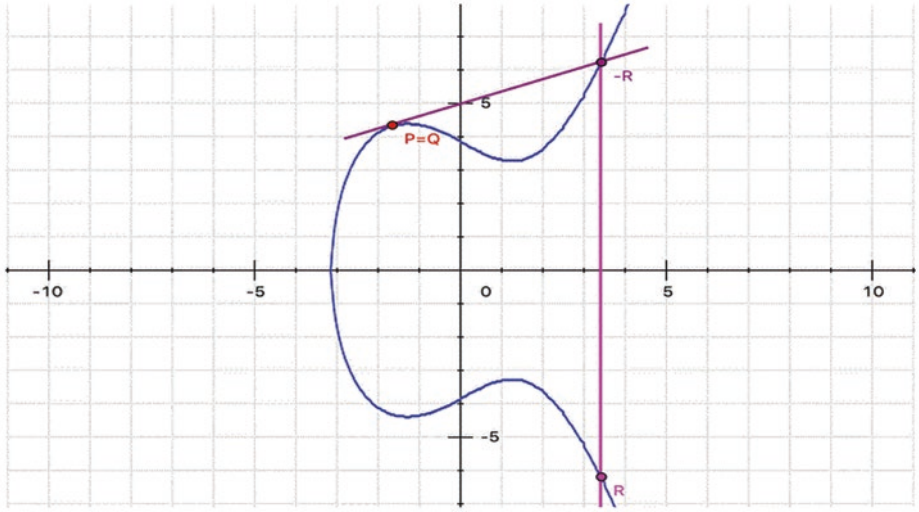

Abb. 4.8 Grafische Darstellung einer Punktdopplung des Punktes P zu $R = P + P$. (Quelle: eigene Darstellung unter Verwendung von Cryptool [7])

Wir sind nun in der Lage, die Schlüsselübereinkunft mittels ECDH zur verstehen. Der ECDH erfolgt nun in Anlehnung an die ursprüngliche Diffie-Hellman-Schlüsselübereinkunft wie folgt: Alice und Bob wählen jeweils unabhängig voneinander ihr geheimes

$\alpha \in \{2, 3, \ldots, \#E\}$ und $\beta \in \{2, 3, \ldots, \#E\}$. Alice berechnet den Punkt A auf der elliptischen Kurve durch Erzeugung von $A = \alpha G$ und Bob berechnet den Punkt B auf der elliptischen Kurve durch Erzeugung von $B = \beta G$. Beide Parteien übertragen den Punkt A bzw. den Punkt B (Abb. 4.9). Da die Punktaddition eine assoziative Operation ist kann Alice mit dem Erhalt von B den Punkt $T = \alpha B = \alpha(\beta G)$ berechnen und Bob seinerseits mit dem Erhalt von A den Punkt T durch $\beta A = \beta(\alpha G)$ berechnen. Da die Koordinaten x_T und y_T des Punktes T auf der Kurve eine gewisse Abhängigkeit aufweisen, wird nur eine dieser beiden Koordinaten verwendet um hieraus einen gemeinsamen Sitzungsschlüssel k für Alice und Bob abzuleiten. Ansonsten, so die Meinung der Sicherheitsexperten, hätte sich in den auszuhandelnden Sitzungsschlüssel eine gewisse Redundanz eingeschlichen. Alice und Bob erhalten den Sitzungsschlüssel durch Anwendung einer Hashfunktion $h()$ auf die x-Koordinate des Punktes T, also $k = h(x_T)$.

Nutzt man elliptische Kurven für eine Diffie-Hellmann-Schlüsselübereinkunft, so müssen sich beide Parteien vorab über die sogenannten Domänenparameter $dom = \{q, a, b, G, n\}$ verständigt haben. Im Einzelnen sind dies der Modulus q, die Faktoren a und b der Kurvengleichung E, ein Generatorpunkt G auf E, sowie die Ordnung n der zyklischen Untergruppe, die der Generatorpunkt G aufspannen kann. Eine kompakte Darstellung der Abfolge einer ECDH-Schlüsselübereinkunft ist in Abb. 4.9 zu sehen.

Die hier beschriebene ECDH-Schlüsselübereinkunft kommt innerhalb von Bluetooth in der zweiten Pairing-Phase zum Tragen und wird anschließend in der Phase 4 des Pairing-Vorganges zur Berechnung des Sitzungsschlüssels *MACKey* mittels der Funktion $fc()$ verwendet, also $MACKey = fc(k, N_a, N_b, MAC_a, MAC_b)$. Dabei geht der Sitzungsschlüssel k, auch als *DHKey* bezeichnet, aus dem elliptischen Kurvenpunkt T und Anwendung einer Hashfunktion $h()$ hervor, wie oben bereits erörtert. Hinzu kommen die beiden Nonces N_a und N_b sowie die MAC-Adressen der Sicherungsschicht beider Geräte. Diese sind für Alice notiert als MAC_a und für Bob notiert als MAC_b.

4.4.2.2 Aus der Praxis

Um dem Leser ein Gespür für die tatsächliche Dimensionierung der in der Praxis verwendeten Kurvenparameter zu ermöglichen soll dies beispielhaft für die standardisierte Kurve `secp256r1` erfolgen. Filtert man hierzu mittels des Werkzeugs Wireshark nach

Alice				Bob
dom	$\rightarrow$	dom	$\rightarrow$	dom
wähle α				wähle β
$A = \alpha G$	$\rightarrow$	A	$\rightarrow$	
	$\leftarrow$	B	$\leftarrow$	$B = \beta G$
$T = \alpha B$				$T = \beta A$
$k = h(x_T)$				$k = h(x_T)$

Abb. 4.9 Abfolge der ECDH-Schlüsselübereinkunft. (Quelle: eigene Darstellung)

einem Handshake über dem elliptische Kurvenparameter zur Etablierung einer sicheren Verbindung ausgehandelt werden, so findet man recht häufig diesen Kurventyp vor. Definiert wird hierüber eine Kurve $y^2 = x^3 + ax + b$ über dem Körper K_p. Im Einzelnen lauten die Domänenparameter $dom = \{p, a, b, G, n\}$ für die Kurve `secp256r1` in hexadezimaler Notation wie folgt:

$$p = FFFFFFFF\ 00000001\ 00000000\ 00000000\ 00000000\ FFFFFFFF\ FFFFFFFF\ FFFFFFFa$$
$$= FFFFFFFF\ 00000001\ 00000000\ 00000000\ 00000000\ FFFFFFFF\ FFFFFFFF\ FFFFFFFC$$
$$b = \ 5AC635D8\ AA3A93E7\ B3EBBD55\ 769886BC\ 651D06B0\ CC53B0F6\ 3BCE3C3E\ 27D2604B$$

Der Punkt G sowie die Ordnung n in hexadezimaler Notation lauten:

$$G = 04\ 6B17D1F2\ E12C4247\ F8BCE6E5\ 63A440F2\ 77037D81\ 2DEB33A0\ 4A13945\ D898C296$$
$$4FE342E2\ FE1A7F9B\ 8EE7EB4A\ 7C0F9E16\ 2BCE3357\ 6B315ECE\ CBB64068\ 37BF51F5$$
$$n = FFFFFFFF\ 00000000\ FFFFFFFF\ FFFFFFF\ BCE6FAAD\ A7179E84\ F3B9CAC2\ FC632551$$

Beispielsweise entspricht p dem Wert

$$2^{224}\left(2^{32} - 1\right) + 2^{192} + 2^{96} - 1.$$

Damit dürfte für den Leser ein wenig nachvollziehbarer sein, welche Größe tatsächliche Kurvenparameter innehaben, denen ein begründbares Sicherheitsniveau hinsichtlich des ECDLP zugetraut wird oder – sollte ich besser sagen – zugetraut wurde. Auch wenn Kurven wie die elliptische Kurve `secp256r1` noch sehr häufig verwendet werden, so ist dies die absolut untere Grenze, was die Parameterwahl angeht. Oder ein wenig drastischer ausgedrückt, wie es Mathew Green in seinem Blog [13] beschreibt, der die Analysen der Sicherheitsexperten Koblitz und Menezes zur Wahl geeigneter elliptischer Kurven zusammenfasst: Bei 2^{209} aller insgesamt denkbaren 2^{257} P-256-Kurven können Sicherheitsschwächen nicht ausgeschlossen werden. Außerdem hält sich recht hartnäckig das Gerücht, dass für die NSA die von NIST standardisierten elliptischen Kurven mit einer Hintertür ausgestattet seien, so dass die von der Crypto Forum Research Group (CFRG) oder der Internet Engineering Task Force (IETF) vorgeschlagenen Kurven bevorzugt zu verwenden seien. Weitere Details zu Parametern der oben genannten Kurve sowie einer ganzen Reihe weiterer standardisierter elliptischer Kurven können dem Dokument [6] entnommen werden.

4.4.3 Angriffe über ungültige Kurven

Kurz nachdem Bluetooth 2.1 mit dem Secure Simple Pairing (SSP) verfügbar war, wurde im Jahr 2008 ein Angriff auf die Phase 2 des Bluetooth-Pairings, den Schlüsselaustausch, öffentlich. Diesen Angriff, mögliche Gegenmaßnahmen sowie eine Variante des ursprünglichen Angriffs aus dem Jahre 2018 wollen wir nun erörtern. Beide Angriffsarten fallen in die Kategorie der sogenannten *‚Small Subgroup‘*-Angriffe, bei denen der

Angreifer durch die Manipulierung einzelner Protokollnachrichten darauf abzielt, die Ordnung der Gruppe zu reduzieren, deren Elemente zur Erzeugung des Schlüssels verwendet werden.

4.4.3.1 Angriff über ungültiger Kurvenpunkte

Betrachtet man die Schreibweise einer elliptischen Kurve E in der sogenannten Weierstrass-Darstellung, also $y^2 = x^3 + ax + b$, und führt die Berechnung eines Ergebnispunktes $C = (x_C, y_C) = A + B$ zweier Punkte $A = (x_A, y_A)$ und $B = (x_B, y_B)$ auf der Kurve E durch, so erkennt man schnell, dass sowohl bei der Punktaddition, als auch bei der Punktdoppplung der Parameter b der elliptischen Kurvengleichung gar nicht in die Berechnung des Ergebnispunktes $C = (x_C, y_C)$ eingeht. Diesen Umstand macht sich der nachfolgend beschriebene Angriff aus dem Jahre 2008 zunutze, der auf die Berechnung des Schlüssels unter anderem bei Verwendung von *Secure Simple Pairing* (SSP) abzielt. Die grundsätzliche Idee stammt aus dem Jahre 2003 und wurde in [2] von Antipa, Brown, Menezes, Struik und Vanstone für eine Reihe von Protokollen, basierend auf elliptischen Kurven beschrieben. An dieser Stelle begnügen wir uns damit, die Grundidee derartiger Verwundbarkeiten aufzuzeigen und ihre Auswirkung auf den Bluetooth-Standard zu erörtern, um anschließend die Wertigkeit der Maßnahmen zur Eindämmung dieser Verwundbarkeit einordnen zu können.

Interessant hierbei ist, dass der Angreifer nicht notwendigerweise ein klassischer MitM-Angreifer Mallory sein muss. Im Falle von Bluetooth reicht es schon aus, wenn Mallory das Opfergerät zu einem Pairing-Vorgang mittels SSP animiert.

Unser Angreifer Mallory verwendet bewusst eine Reihe von Punkten, die nicht auf derjenigen elliptischen Kurve E liegen, auf deren Domänenparameter $dom = \{q, a, b, G, n\}$ sich beide Parteien im Vorfeld des eigentlichen Pairings verständigt haben. Stattdessen sind die Punkte Elemente anderer elliptischer Kurven E'. Mallory wählt die Kurven E' nun so, dass diese den Aufbau $y^2 = x^3 + ax + b'$ haben, wobei er darauf achtet, dass $b \neq b'$ gilt und b der Parameter der ursprünglich vereinbarten Kurve E ist. Der Trick dabei: Die Kurve E' ist so gewählt, dass die Ordnung p der zyklischen Gruppe sehr viel kleiner ist als n. Oder ein wenig konkreter, so wie dies in [2] geschildert ist: Mallory sendet so lange Nachrichten mit derartigen geformten Punkten, bis er auf diesem Wege ein System von Kongruenzbeziehungen erhalten hat, das $N = \prod_{(i=1)}^{l} p_i > n^2$ erfüllt.

Unter der Vorbedingung, dass die Opfer Alice (oder Bob) mit Durchführung des ECDH bei Eingang eines elliptischen Kurvenpunktes B bzw. A typischerweise nicht prüfen, ob dieser Punkt tatsächlich auf der Kurve E liegt, kann der Angreifer Mallory nun auch Punkte Q der obigen Bauart an das Opfer senden. Alice antwortet, indem sie den empfangenen Punkt multipliziert mit dem eigenen geheimen Skalar α und daraus wie gehabt den Punkt T erhält. Durch Anwendung einer Hashfunktion auf die x-Koordinate von T erhält Alice den gemeinsamen Schlüssel k. Diesen nutzt sie, um in der Folge Nachrichten mittels eines Message Authentication Codes zu authentifizieren und zu versenden.

Mallory wendet seinerseits denselben Algorithmus auf die Punkte $T_1 = Q, T_2 = 2Q$, …, $T_p = pQ$ an. Das ist mit überschaubarem Aufwand möglich, weil p klein ist. Aus diesen Punkten $T_j (j = 1, \ldots, p)$ berechnet Mallory nun mögliche Schlüssel $k_1 = h(x_{T1})$, $k_2 = h(x_{T2})$, …,$k_p = h\left(x_{Tp}\right)$. Mit dem Erhalt der Nachricht m und einer Signatur s von Alice kann Mallory die Message Authentication Codes $s_1 = MAC_{k1}(m)$, $s_2 = MAC_{k2}(m)$, …, $s_p = MAC_{kp}(m)$ berechnen und mit der Signatur s von Alice vergleichen. Dasjenige j, für das $s_j = s$ gilt, gehört zum Punkt $T_j = jQ$ und entspricht dem Punkt $T = \alpha Q$ von Alice. Dies bedeutet, dass der private Schlüssel α von Alice der Kongruenzbeziehung $\alpha \equiv j (mod\ p)$ genügt.

Dieser am Beispiel der elliptischen Kurve E' beschriebene Angriff wird nun von Mallory für eine Reihe von Kurven $E_i (i = 1, \ldots, l)$ ausgeführt, wobei auch hier wieder gilt, dass alle $E_i \neq E$. Mallory sendet also Punkte $Q_i \in E_i (i = 1, \ldots, l)$ an Alice und kann dadurch mit der Zeit ein System von gleichzeitigen Kongruenzen aufbauen:

$$\begin{aligned} \alpha &\equiv j_1 \ (\text{mod}\ p_1) \\ \alpha &\equiv j_2 \ (\text{mod}\ p_2) \\ &\vdots \\ \alpha &\equiv j_l \ (\text{mod}\ p_l) \end{aligned}$$

Dabei ist p_i die Ordnung der zyklischen Gruppe die durch die elliptische Kurve E_i erzeugt wurde. Die Anzahl l aller Kongruenzen kann wie folgt bestimmt werden: Es werden solange weitere Kurven E_i gewählt und Punkte Q_i gesendet, bis die Ungleichung $N = \prod_{i=1}^{l} p_i > n^2$ erstmalig erfüllt ist.

Die gerade geschilderte Abfolge ist noch einmal in einer komprimierten Darstellung in Abb. 4.10 aufgeführt. Dabei sind diejenigen Berechnungen und Nachrichtenübertragungen, die aufgrund des eigentlichen ECDH nach wie vor durch das ahnungslose Opfer erfolgen, aber für das Gelingen des geschilderten Angriffs gänzlich unerheblich sind, in der Farbe Grau angedeutet.

Mallory				Alice
$dom(E) = (q, a, b, G, n)$				$dom(E) = (q, a, b, G, n)$
solange $(\prod_{i=1}^{l} p_i < n^2)$\{				wähle α
wähle $Q_i \in E_i \neq E$	→	Q_i	→	
$T_j = jQ$	←	A	←	$A = \alpha G$
				$T_j = \alpha Q_i$
				$k = h(x_{Ti})$
prüfe $s_j ? = s$	←	m, s	←	$s = MAC_k(m)$
\}				
CR				
$\alpha = Gauss(CR)$				

Abb. 4.10 Protokollnachrichten der ECDH-Schlüsselübereinkunft bei Angriff mit ungültigen Kurvenpunkten $Q_i \in E_i$ und $E_i \neq E$. (Quelle: eigene Darstellung)

Dieser Angriff ist für Mallory mit einem vertretbaren Aufwand durchführbar. Antipa und Co-Autoren haben innerhalb ihrer Ausarbeitungen gezeigt, dass Mallory für die vom Standardisierungsgremium NIST empfohlene Kurven *E,* im besten Falle $l = 61$ Punkte zu senden hat, um den geschilderten Angriff erfolgreich durchzuführen.

Das so erzeugte System von Kongruenzen lässt sich beispielsweise unter Anwendung des Gauß-Algorithmus für den chinesischen Restsatz *(CR)* lösen und damit kann der geheime Schlüssel α des Opfers Alice berechnet werden:

$$\begin{aligned} \alpha &\equiv j_1 \pmod{p_1} \\ \alpha &\equiv j_2 \pmod{p_2} \\ &\vdots \\ \alpha &\equiv j_l \pmod{p_l} \end{aligned}$$

Mallory sieht sich also mit einer Reihe ganzer Zahlen p_i konfrontiert, welche zueinander paarweise teilerfremde natürliche Zahlen bilden, wenn er auf den geheimen Skalar α von Alice schließen möchte.

Wie wir in Abschn. 2.3 bereits erfahren haben, kann Mallory hierzu den Gauß-Algorithmus anwenden. Er ermittelt den Wert für den geheimen Skalar α von Alice durch Berechnen der Summe

$$\sum_{i=1}^{l} j_i \, N_i M_i \bmod \mathrm{n}$$

Der geheime Skalar α ist dann eindeutig Modulo $n = p_1 p_2 \ldots p_l$. Hierbei ist $N_i = n/p_i$ und $M_i = N_i^{-1} \; mod \; p_i$.

Bitte beachten Sie, dass die hier gegebene Beschreibung einen tatsächlich erfolgreich durchzuführbaren Angriff abstrahiert. Insbesondere ist für eine praktische Umsetzung des Angriffs zu unterscheiden, welche konkrete ECDH-Variante in einer Implementierung umgesetzt ist und zum Einsatz kommt.

Der obige Angriff funktioniert immer dann, wenn die privaten Schlüssel α und β von Alice und Bob nicht geändert werden, so wie es bei den Bluetooth-SSP-Implementierungen umgesetzt war. Dies wurde dann in der Bluetooth-Spezifikation v5.0 [5] geändert, allerdings, man höre und staune, erst im Jahre 2016. Hier wird nun gefordert, den privaten Schlüssel von Zeit zu Zeit, in Abhängigkeit von der Anzahl der eingegangenen Pairing-Anfragen zu ändern. Offensichtlich sollte durch diese Maßnahme die Bedingung $\prod_{i=1}^{l} p_i > n^2$ unerfüllbar werden. Denn durch eine derartige Maßnahme wird eine nicht ausreichende Anzahl an Kongruenzbeziehungen erzwungen, mit der die Anwendung des Gauß-Algorithmus zu keiner eindeutigen Lösung mehr führt.

Machen Sie sich bitte klar: Angriffe auf ECDH-Protokollvarianten, bei denen der Angreifer Punkte $Q_i \in E_i (i = 1, \ldots, l)$ mit $E_i \neq E$ versendet würden ins Leere laufen und nicht mehr funktionieren, würde die Gegenstelle tatsächlich überprüfen, ob ein eingehender Punkt Q_i zu den vereinbarten Domänenparametern $dom(E) = (q, a, b, G, n)$

passt. Laut [2] ist ein Punkt Q_i immer dann gültig, wenn gleichzeitig alle der nachfolgend aufgeführten Bedingungen erfüllt sind:

1. $Q_i \neq \infty$
2. Die Koordinaten x_{Qi} und y_{Qi} sind Elemente des endlichen Körpers $\boldsymbol{K}_q$
3. Q_i gehört zu E
4. $nQ_i = \infty$ (n ist die Ordnung zu G)

Da derartige Überprüfungen jedoch in den meisten umgesetzten ECDH-Implementierungen gar nicht oder nur teilweise erfolgten, war die hier beschriebene Verwundbarkeit bis zu einer Anpassung der Spezifikation v5.0 sehr real.

▶ Der erste *‚Small Subgroup‘*-Angriff mittels ungültiger Kurvenpunkte $Q_i \in E_i \neq E (i = 1, \ldots, l)$ und kleiner Ordnungen p_i ermöglicht Mallory bei Bluetooth und einem Pairing mittels SSP durch Lösen des chinesischen Restsatzes mit Kongruenzbeziehungen $\alpha \equiv j(modp)$ auf den geheimen Skalar α von Alice beim ECDH zu schließen. Die Verwundbarkeit resultiert aus einer ursprünglich **fehlerhaften Spezifikation,** die auch die Bearbeitung von Punkten $Q_i \in E_i$ erlaubt, die nicht der ursprünglich vereinbarten Kurve E angehören. Als Gegenmaßnahme enthalten neuere Spezifikationen die Forderung, die geheimen Skalare α und β von Alice und Bob häufig zu ändern. Die Änderungen der Skalare α und β müssen erfolgt sein bevor der Angreifer l Punkte Q_i versenden kann. Der Parameter l ergibt sich aus der Bedingung $\prod_{i=1}^{l} p_i > n^2$.

4.4.3.2 Angriff über geänderte Punktkoordinaten

Der Angriff von Biham und Neumann aus dem Jahre 2018 ist nur eine weitere Variante aus der Kategorie der Angriffe über *‚ungültige Kurven‘,* die eigentlich schon seit längerer Zeit bekannt sind und die wir im vorherigen Abschnitt beschrieben haben. In der konkreten Ausgestaltung beruht diese Verwundbarkeit nun auf der Beobachtung, dass nur die x-Koordinate eines Punktes A oder B beim ECDH-Nachrichtenaustausch jeder Partei während der Bluetooth-Authentifizierung eingeht und der hierdurch abgeleitete Schlüssel auf eine kleinere Untergruppe beschränkt wird. Hieraus resultieren Erfolgsaussichten von 25 % oder 50 % für den Angreifer in Abhängigkeit davon, wie viele der ECDH-Nachrichten der Angreifer abfangen und modifizieren kann. An dieser Stelle sei nur der *semipassive* Angriff mit einer Erfolgswahrscheinlichkeit von 25 % beschrieben, der *vollständig aktive* Angriff mit einer Erfolgswahrscheinlichkeit von 50 % ist ausführlich in [3] dargestellt und dort analysiert. Das prinzipielle Angriffsmuster lässt sich jedoch auch sehr gut anhand der Erörterung des *semipassiven* Angriffs nachvollziehen. Die genannten Erfolgsaussichten beziehen sich auf genau einen Angriff der geschilderten Art. Da es dem Opfergerät in aller Regel aber nicht gelingen dürfte, zu erkennen, dass ein Angriff stattgefunden hat, kann der Angriff so oft initiiert werden bis er tatsächlich

erfolgreich ist, sodass die tatsächliche Erfolgswahrscheinlichkeit wohl mit 100 % zu beziffern ist.Für beide Angriffsformen ist die Beobachtung essenziell, dass die Ergebnisse der Punktoperationen unabhängig sind von der Wahl des konkreten Parameters b in der zugrunde gelegten Kurve. Dies bedeutet, dass unabhängig davon, ob die Kurve $E : y^2 = x^3 + ax + b$ oder $E' : y^2 = x^3 + ax + b'$ lautet, das Ergebnis jeder Punktaddition oder auch Punktdopplung eines Punktes auf E oder E' das gleiche ist. Des Weiteren muss man für ein weiterführendes Verständnis dieses Angriffs damit vertraut sein, dass das Inverse A^{-1} eines Punktes $A = (x_A, y_A)$ sich durch seine Spiegelung an der x-Achse ergibt, also $A^{-1} = (x_A, -y_A)$. Für einen Punkt $A' = (x_A, 0)$ ist das Inverse A'^{-1} dieses Punktes somit wiederum der Punkt A', also $(x_A, 0)$. Eine Schwierigkeit besteht noch: Ändert ein Angreifer einen Punkt $A = (x_A, y_A)$ in einen Punkt $A' = (x_A, 0)$, so befindet sich dieser Punkt nicht mehr auf der Kurve E auf welche sich Alice und Bob ursprünglich durch die Übergabe der Domänenparameter $dom(E)$ verständigt haben. Man kann E' jedoch so konstruieren, dass der Punkt A' tatsächlich auf der Kurve E' liegt. Dies gelingt Mallory, indem er den Parameter b' der Kurve anpasst. Biham und Neumann geben in ihrer Ausarbeitung hierfür den Wert $b' \equiv -x_A^3 - ax_A (mod\ q)$ an. Eine solche Kurve E', die Mallory mit besagtem b' aus der ursprünglichen Kurve E' konstruiert hat, und der Punkt $A' = (x_A, 0)$ sind in Abb. 4.11 dargestellt. Über solch ein Konstrukt bildet $A' = (x_A, 0)$ nun den Generatorpunkt einer Gruppe, die aus genau zwei Elementen besteht: dem Punkt A' selbst sowie dem unendlichen Punkt ∞. Denn immer dann, wenn man A' mit einem ungeraden Skalar n_u multipliziert, so ist das Ergebnis $A' = n_u\, A'$, also wiederum der Punkt $A' = (x_A, 0)$. Multipliziert man A' hingegen mit einem geraden Skalar n_g, so ist $\infty = n_g\, A'$.

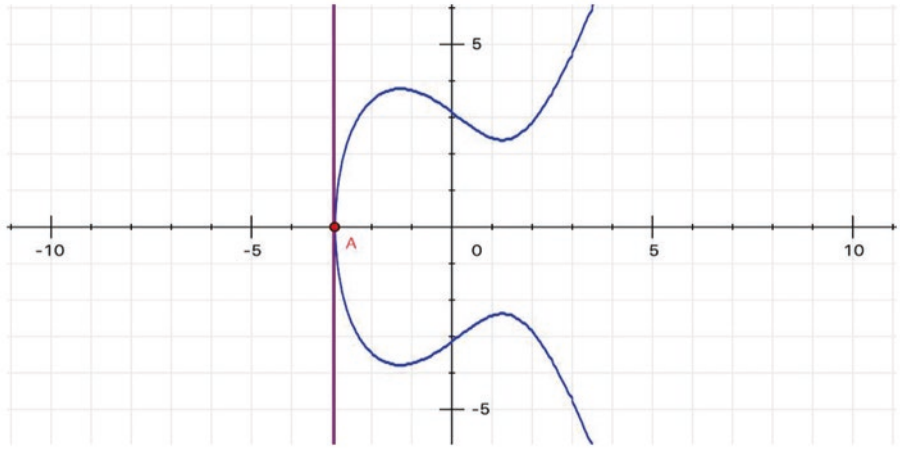

Abb. 4.11 Grafische Darstellung eines Punktes $A' = (x_A, 0)$ auf E mit $b' \equiv -x_A^3 - ax_A (mod\ q)$. (Quelle: eigene Darstellung unter Verwendung von Cryptool [7])

Damit haben wir den Grundgedanken dieser von den Sicherheitsexperten Biham und Neumann vorgestellten Angriffsform skizziert: Mallory ändert die beim ECDH zwischen den Opfern Alice und Bob gesendeten Punkte $A = (x_A, y_A)$ und $B = (x_B, y_B)$ in die Punkte $A' = (x_A, 0)$ und $B' = (x_B, 0)$ mit dem Ziel, aufgrund der extrem reduzierten Gruppengrößen, die nun zur Anwendung kommen, Rückschlüsse auf den ausgehandelten Schlüssel zu erhalten. Dies ist ihm möglich, da die Bluetooth-Spezifizierung vorsieht, allein die x-Koordinaten der Punkte einer Authentizitätsprüfung zu unterziehen, nicht aber deren y-Koordinaten. Dieses Vorgehen hilft Mallory enorm. Allerdings ist Mallory damit noch nicht am Ziel seiner Wünsche. Denn Mallory hat keinen Einfluss auf die Wahl der beiden Punkte $A = (x_A, y_A)$ und $B = (x_B, y_B)$ und kann, anders als die Werte y_A und y_B, die Werte x_A und x_B eben nicht unerkannt ändern. Dies bedeutet nun, dass in den allermeisten Fällen die von Mallory manipulierten Punkte $A' = (x_A, 0)$ und $B' = (x_B, 0)$ nicht aus der gleichen Gruppe stammen dürften. Jede weiter durchgeführte ECDH-Berechnung auf Basis dieser so geänderten Punkte führt daher bei Alice und Bob sicherlich nicht zum gleichen Ergebnispunkt T aus dem anschließend der gemeinsame symmetrische Schlüssel abzuleiten wäre. Ist allerdings der geheime Skalar α, den Alice gewählt hat, gerade und gleichzeitig der geheime Skalar β, den Bob gewählt hat, ebenfalls gerade, so ergibt sowohl für Alice als auch für Bob $\infty = \alpha_g B' = \beta_g A'$. Unter diesen Umständen ist auch das Ergebnis für T auf der Seite von Alice und von Bob identisch. Denn es lautet $T = \infty$. Dies ist bei 1/4 aller Angriffe der Fall und genau dann kann Mallory auf den gemeinsamen Schlüssel schließen. Die Erfolgsaussichten für den von Biham und Neumann vorgestellten semipassiven Angriff lassen sich daher wie in Abb. 4.12 zusammenfassen.

Die Abfolge der Nachrichten für die semipassive Variante des Angriffs über *‚ungültige Kurven‘* im Erfolgsfall für Mallory ist in der Abb. 4.13 noch einmal aufbereitet. Als notwendige Voraussetzung für ein erfolgreiches Gelingen eines derartigen Angriffs müssen sowohl Alice als auch Bob jeweils gerade Skalare α_g und β_g als Geheimnisse verwenden.

		Geheimnis Bob (Skalar **β**)	
		β gerade (β_g)	β ungerade (β_u)
Geheimnis Alice Skalar **α**	α gerade (α_g)	Mallory erfolgreich ($T = \infty$.)	Mallory erfolglos
	α ungerade (α_u)	Mallory erfolglos	Mallory erfolglos

Abb. 4.12 Erfolgsaussichten des Angriffs auf ECDH über geänderte y-Koordinaten in Abhängigkeit von der Beschaffenheit der geheimen Skalare α und β. (Quelle: eigene Darstellung in Anlehnung an [3])

Alice		Mallory		Bob
$dom(E)$	$\rightarrow$	$dom(E)$	$\rightarrow$	$dom(E)$
α_g				β_g
$\alpha_g G = A = (x_A, y_A)$	$\rightarrow$	$A = (x_A, y_A) \mid A' = (x_A, 0)$	$\rightarrow$	
	$\leftarrow$	$B' = (x_B, 0) \mid B = (x_B, y_B)$	$\leftarrow$	$\beta_g G = B = (x_B, y_B)$
$T = \alpha_g (x_B, 0) = \infty$		$T = \infty$		$T = \beta_g (x_A, 0) = \infty$

Abb. 4.13 Protokollabfolge der semipassiven-Variante des Angriffs über ‚ungültige Kurven'. In dem dargestellten Fall ist der Angriff erfolgreich da sowohl das Opfer Alice als auch das Opfer Bob gerade Skalare α_g und β_g verwenden. (Quelle: eigene Darstellung)

Man muss sich klarmachen, dass ein derartiger MitM-Angriff zu einem anderen Resultat führt als ein herkömmlicher MitM-Angriff, bei dem sich der Angreifer gegenüber Alice als Bob und gegenüber Bob als Alice ausgibt. Bei letzterem Szenario hält der Angreifer nach erfolgreichem Angriff zwei Sitzungsschlüssel, den Schlüssel K_{AM} für die Kommunikation mit Alice und den Schlüssel K_{BM} für die Kommunikation mit Bob. Im Falle eines erfolgreichen ‚Ungültige Kurve'-MitM-Angriffs erfährt Mallory genau einen Sitzungsschlüssel (Abb. 4.14). Damit muss der Angreifer zum Zeitpunkt der Datenübertragung nicht als Proxy für jedwede Kommunikation zwischen Alice und Bob verfügbar sein, sondern kann die Daten beispielsweise erst einmal mitschneiden und später auswerten.

Doch wie konnte es überhaupt zu einer derartigen Schwachstelle kommen? Die wesentliche Designschwäche, die den geschilderten Angriff ermöglicht, mutet bizarr an: So erwartet der Bluetooth-Standard ausschließlich eine Authentifizierung der *x*-Koordinaten, sodass die *y*-Koordinaten der übermittelten elliptischen Kurvenpunkte *A* und *B* während des Pairings tatsächlich nicht authentifiziert werden. Dies wird unterlassen, obwohl der Standard v4.2 aus dem Jahre 2014 schon Gegenmaßnahmen gegen allgemeine und im vorherigen Abschnitt geschilderte Angriff über ‚ungültige Kurven' enthält. Diese können die geschilderte verfeinerte Variante der *‚Small Subgroup'*-Verwundbarkeit jedoch keineswegs beheben: Vorgeschrieben ist lediglich die Erneuerung des ECDH-Schlüssels mit jedem Pairing-Versuch. Dies führt zur Unterbindung vom *‚Small Subgroup'*-Angriffen, so wie wir sie im vorherigen Abschnitt beschrieben haben,

	Alice	MitM	Bob
herkömmlicher MitM	K_{AM}	K_{AM}, K_{BM}	K_{BM}
‚Ungültige Kurve'-MitM	K_{AB}	K_{AB}	K_{AB}

Abb. 4.14 Schlüsselverteilung nach erfolgreichem herkömmlichem MitM- und ‚Ungültige Kurve'-MitM-Angriff. (Quelle: eigene Darstellung)

nicht jedoch zu denjenigen, wie wir sie in diesem Abschnitt erörtert haben. Die einfachste Gegenmaßnahme, nämlich die Authentifizierung der *y*-Koordinate neben der *x*-Koordinate, sieht der Standard hingegen nicht vor.

▶ Die zweite *‚Small Subgroup'*-Verwundbarkeit auf das *LE Secure Connections* und *Secure Simple Pairing* bei Bluetooth beruht auf der Tatsache, dass beim Aushandeln eines gemeinsamen Geheimnisses mittels ECDH nur die *x*-Koordinate eines Punktes in die Schlüsselberechnung eingeht und auch nur diese während des Pairings authentifiziert und geprüft wird. Dies kann von einem MitM-Angreifer genutzt werden, indem die *y*-Koordinaten der Punkte *A* und *B* auf null gesetzt werden und damit oftmals die weitere Berechnung des Sitzungsschlüssels vorhersagbar wird, und zwar immer dann, wenn der gemeinsame Punkt $T = \infty$ ist. Die Verwundbarkeit resultiert aus einer schon **bizarr anmutenden Ausgestaltungsschwäche der Bluetooth-Spezifikation,** bei der nur Teile der übertragenen Punkkoordinaten authentifiziert werden.

Allerdings weisen die Sicherheitsexperten Biham und Neumann auch darauf hin, dass der Aufbau und die praktische Durchführung eines echten MitM-Angriffs zwischen zwei Bluetooth-Geräten mit ECDH aktuell nicht möglich ist. So wurde lediglich die prinzipielle Machbarkeit dieses Angriffs verifiziert, indem ein Bluetooth USB Dongle verwendet wurde, auf dem die *y*-Koordinaten der elliptischen Kurvenpunkte direkt genullt worden sind. Ein tatsächlicher MitM-Angriff fand somit nicht statt. Es wurden keine zwischen zwei Bluetooth-Geräten übertragenen Rahmen mitgeschnitten bzw. modifiziert. Jedoch ermöglicht es das Bluetooth USB Dongle (CY5677 CySmart) Verbindungen herzustellen, zu debuggen und den Angriff nachzustellen.

Weitere Hürden, die die Praxisrelevanz des Angriffs zumindest einschränken, sind:

1. Der Angriff ist in seiner geschilderten Form ausschließlich bei einem erstmaligen Pairing-Vorgang durchführbar. Nur unter diesen Umständen haben sich die Geräte noch nicht auf einen Langzeitschlüssel verständigt der bei allen nachfolgenden Pairing-Vorgängen eingeht.
2. Bluetooth verwendet ein Frequenzsprungverfahren. Hierbei wird nach einem pseudozufälligen Muster 1600 Mal pro Sekunde zwischen 79 möglichen Frequenzen gewechselt. Auch wenn dieses pseudozufällige Muster, das vom Mastergerät vorgegeben wird, keine wirkliche Sicherheit bietet, so ist es doch für den geschilderten Angriff eine Hürde, die Mallory überwinden muss. Gemeinsam mit der Tatsache, dass bei Bluetooth ein zu sendendes Signal zerlegt und über die gesamte Breite des Frequenzbereiches gespreizt wird, erhöht dies für Mallory den Aufwand bzw. die technischen Voraussetzungen.
3. Neumann und Biham weisen in ihrer Arbeit darauf hin, dass die praktische Umsetzung des MitM-Angriffs erleichtert wird, wenn die *x*-Koordinate und die *y*-Koordinate in

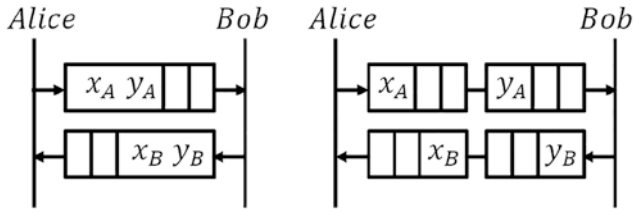

Abb. 4.15 Übertragung der Punktkoordinaten in Bluetooth-Paketen bei ECDH in Abhängigkeit von der zugrundeliegenden Schlüssellänge: links für Schlüssellängen <256 Bit, rechts für Schlüssellängen ≥256 Bit. (Quelle: eigene Darstellung)

verschiedenen Rahmen transportiert werden. Paradoxerweise ist dies der Fall bei Kurven mit einem eigentlich höheren Sicherheitsniveau. So werden die Punktkoordinaten bei Kurven mit einer Schlüssellänge ab 256 Bit in zwei getrennten Rahmen übertragen, während Kurven, die eine kleinere Schlüssellänge bedienen, die Punktkoordinaten in ein und demselben Rahmen übertragen. Die Abb. 4.15 veranschaulicht diesen Sachverhalt, ohne dass hierbei die genauen Größenverhältnisse für die Bereiche Zugriffscode (72 Bit), Paketkopf (54 Bit) und Nutzdaten (0–2745 Bit) berücksichtigt wären.

4.4.4 Anmerkungen zu Seitenkanalangriffen auf ECC und ECDH

Oftmals können Ressourcen oder Funktionalitäten in IT-Systemen, die ursprünglich niemals als Kommunikationskanal gedacht waren, dennoch findig genutzt werden, um Informationen aus dem System abfließen zu lassen. Solche verdeckten Kanäle *(covert channel)* oder aber auch Seitenkanäle *(side channel)* sind beispielsweise modellierbar, indem ein Angreifer das Zeitverhalten oder andere unvermeidbar nach außen dringende Verhaltensmuster bei der Ausführung bzw. dem Zugriff auf geteilte Ressourcen nutzt. Dies ist natürlich immer dann äußerst beunruhigend, wenn auf diese Weise Schlüsselmaterial über einen derartigen Seitenkanal abfließen kann. Mögliche Seitenkanalangriffe unter Ausnutzung der Eigenschaften von ECC-Implementierungen bzw. ECDH bilden hier keinerlei Ausnahme (Abb. 4.17). Daher wollen wir uns an dieser Stelle zumindest einen kompakten Einblick in diese Art von Verwundbarkeiten auf mobilen Systemen verschaffen, auch wenn diese Angriffsart ein generelles Problem für alle IT-Komponenten ist, unabhängig ob diese nun als mobil oder stationär einzuordnen wären.

Durch den Angriff über ‚Ungültige Kurven' auf LE Secure Connections treten andere seit längerem etablierte Angriffe auf ECDH, bei denen der Angreifer physischen Zugriff auf eines der Geräte haben muss, in den Hintergrund. Der Vollständigkeit halber sei an dieser Stelle auch auf diese Angriffsformen eingegangen. Sie finden ihre Anwendung nicht allein bei Bluetooth in Verbindung mit ECDH. Sie eignen sich für alle Protokolle, die einen Beschleunigungsalgorithmus wie *Square-and-Multiply* zur schellen

Exponentiation für RSA einsetzen, oder bei ECC-basierten Verfahren, in denen als Beschleunigungsalgorithmus eine Variante des *Double-and-Add*-Algorithmus verwendet wird. Solche Beschleunigungsalgorithmen gehen zwingend in die Umsetzung asymmetrischer kryptografischer Verfahren ein, um die Anzahl der erforderlichen Rechenoperationen gegenüber herkömmlichen Schulbuch-Ansätzen auf ein praktikables Maß zu reduzieren. So ist im Falle von ECC und damit auch bei der Verwendung von ECDH die Berechnung $dP = T$ teuer. Diese sogenannte Skalarmultiplikation einer Ganzzahl d mit einem elliptischen Kurvenpunkt P kann mithilfe des *Double-and-Add*-Algorithmus beschleunigt werden. Hierzu stelle man sich den Wert d in seiner binären Darstellung vor, also $d_t 2^t + d_{(t-1)} 2^{(t-1)} + \ldots + d_1 2^1 + d_0 2^0$ wobei gilt, dass $d_t = 1$, und betrachte jedes Bit beginnend mit dem *Most Significant Bit* (MSB), also von ‚links nach rechts'. Ist das so erhaltene Bit eine 1, so wendet der Algorithmus die Operation Punktdopplung und darauf folgend die Punktaddition an. Ist das aktuelle Bit in der Binärdarstellung des Skalars d hingegen eine 0, so wendet der *Double-and-Add*-Algorithmus ausschließlich die Operation Punktdopplung an. Eine Ausnahme bildet das führende Bit, das immer ein 1er-Bit ist. Eine Variante des *Double-and-Add*-Algorithmus in Pseudocode-Darstellung ist Abb. 4.16 zu entnehmen. Durch diesen Beschleunigungsalgorithmus kann die Gesamtanzahl der Punktoperationen auf $1{,}5t$ beschränkt werden, wobei der Wert t die Anzahl der Bits der Ganzzahl d in ihrer binären Darstellung beschreibt und man für eine Abschätzung der Laufzeit annimmt, dass die 1er und der 0er-Bits in der Binärdarstellung des Skalars im Schnitt gleich häufig auftreten. Man mache sich klar: Nur über diese Form der Beschleunigung können überhaupt Skalare der Größenordnung 163 Bit und größer verarbeitet werden, so wie dies für ein begründbares Sicherheitsniveau erforderlich ist.

Allerdings hat sich herausgestellt, dass genau solche Beschleunigungsalgorithmen, und im Falle von ECDH der *Double-and-Add*-Algorithmus, eine ideale Grundlage für Seitenkanalangriffe bieten. Ist der Angreifer in der Lage den am Gerät anfallenden Stromverbrauch über die Zeit zu messen, so sind auf diese Weise Rückschlüsse auf die Reihenfolge der ausgeführten Punktdopplungen und Punktadditionen möglich. Ein kurzes am Oszilloskop aufgezeigtes charakteristisches Intervall lässt auf die alleinige Durchführung einer Punktdopplung schließen, während ein längeres Intervall auf die Durchführung einer Punktdopplung mit anschließender Punktaddition hindeutet. Auf

```
Eingabe: E, P, d in Binärnotation notiert als d_t2^t + d_{t-1}2^{t-1} + … + d_12^1 + d_02^0 und d_t=1
Ausgabe: dP
Q ← 0
FOR i FROM t TO 0 DO
   Q ← PunktDopplung(Q)
   IF d_i = 1 THEN
      Q ← PunktAddition(Q,P)
return Q
```

Abb. 4.16 Pseudocode zur Berechnung von dP in einer Variante des *Double-and-Add*-Algorithmus. (Quelle: eigene Darstellung in Anlehnung an [8])

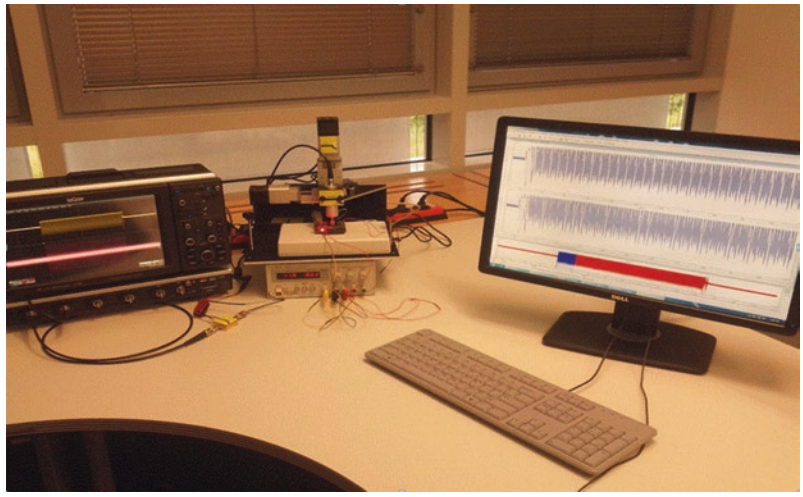

Abb. 4.17 Versuchsaufbau für einen Seitenkanalangriff auf eine Implementierung des *Double-and-Add*-Algorithmus am IHP in Frankfurt-/Oder in der Arbeitsgruppe von Peter Langendörfer. (Quelle: eigene Darstellung)

diese Weise lässt sich also die Binärdarstellung des Skalars d nachvollziehen, was nichts anderes bedeutet, als dass der Angreifer hierdurch in die Lage versetzt ist, den privaten Schlüssel zu ermitteln, oder im Falle von ECDH zur Aushandlung eines gemeinsamen Schlüssels die Geheimnisse $\alpha \in \{2, 3, \ldots, \#E\}$ bzw. $\beta \in \{2, 3, \ldots, \#E\}$ von Alice und Bob in Erfahrung bringen kann.

Genau aus diesem Grunde widmen sich Sicherheitsexperten zur Abwehr derartiger auf differenzieller Stromanalyse beruhender Seitenkanalangriffe erweiterten Formen der Berechnung von dP. Die Sicherheitsforscherin Zoyta Dyka [10] und Co-Autoren haben unterschiedliche Implementierungen untersucht, die eine aussagekräftige Stromanalyse dieser Skalarmultiplikation erschweren, ohne dabei die eigentliche Effizienz der Berechnung von dP signifikant zu verlangsamen. Eine untersuchte Möglichkeit hierfür ist es, über eine gezielte Parallelverarbeitung einzelner Registeroperationen nicht nur die Effizienz der Berechnung zu steigern, sondern gleichzeitig dafür zu sorgen, dass die charakteristischen Stromverläufe für Punktdopplungen und Punktadditionen in einem hohen Maße verschleiert werden. Eine sehr einfache Ausgestaltung wäre es beispielsweise, den *Double-and-Add*-Algorithmus aus Abb. 4.16 so zu modifizieren, dass auch für alle Bits, für die $d_t = 0$ ist, die Operation `PunktAddition(Q,P)` ebenfalls durchgeführt wird. Für diese Fälle darf dann natürlich keine Zuweisung „`Q` ←“ mehr erfolgen, damit das Ergebnis für die eigentliche Punktberechnung von dP nicht verfälscht wird. Eine derartige Härtung gegenüber einem Seitenkanalangriff der geschilderten Art hätte dann einen Schnelligkeitsverlust von $t/2$ mal die CPU-Zyklen für die Operation `PunktAddition` zur Folge. Ergo würden nun insgesamt $2t$ Punktoperationen anstelle der ursprünglichen $1{,}5t$ Punktoperationen benötigt werden. In konkreten Zahlen

ausgedrückt: Unter Verwendung der in Abschn. 4.4.2.2 aufgeführten Kurve secp256r1 wären hierfür also immerhin ≈128 zusätzliche Punktoperationen erforderlich.

▶ Die geschilderte Seitenkanal-Verwundbarkeit auf ECDH beruht auf der Tatsache, dass der Beschleunigungsalgorithmus *Double-and-Add* zur schnellen Berechnung von Punktoperationen von einem Angreifer zur Erlangung des geheimen Skalars α (bzw. β) genutzt werden kann. Hierzu muss sich das Gerät in der Hand des Angreifers befinden, und zwar so, dass dieser den Stromverbrauch zur Ausführungszeit messen kann. Ein kurzzeitig hoher Stromverbrauch weist auf eine Punktdopplung mit anschließender Punktaddition hin, ein schwächerer Stromverbrauch auf eine alleinige Punktdopplung. Der erste Fall deutet auf ein 1er-Bit im Skalar hin, der zweite auf ein 0er-Bit. Auf diese Weise offenbart die Ausführung des Double-and-Add-Algorithmus den geheimen Schlüssel. Die Verwundbarkeit resultiert aus der Verwendung des Beschleunigungsalgorithmus. Sie ist **kein Spezifikationsfehler** von ECDH, sondern, wenn man so will, **eine Implementierungsschwäche.** Eine gezielte Parallelverarbeitung soll die Beschleunigungsvorteile beibehalten und gleichzeitig die Verwundbarkeit über diesen Seitenkanal eindämmen.

4.5 Das „blutende“ Bit

Im Jahre 2018 wurden weitere Schwachstellen auf Bluetooth aufgedeckt und veröffentlicht [1]. Wiederum haben sich hierbei Sicherheitsforscher der Firma Armis hervorgetan. Konkret wurde eine dieser Schwachstellen mit den Namen „blutendes“ Bit auf der Sicherheitskonferenz *Blackhat Europe* im Jahre 2018 von den Sicherheitsexperten Ben Seri und Dor Zusman vorgestellt. Die von ihnen entdeckten Verwundbarkeiten betreffen eine Vielzahl von mobilen, Bluetooth-fähigen Geräten, die mit bestimmten Bluetooth-Low-Energy-(BLE-)-Chips ausgestattet sind.

Im Einzelnen handelt es sich um zwei Verwundbarkeiten der Form *Remote Code Execution,* wobei eine der beiden offengelegten Verwundbarkeiten zusammen mit der Funktionalität zum Updaten der Firmware über die Luftschnittstelle auszunutzen ist. Wir konzentrieren uns im Rahmen dieses Buches jedoch einzig auf die erste der beiden Schwachstellen welche von den Sicherheitsexperten Ben Seri und Dor Zusmann vorgestellt worden sind. Befindet sich ein Angreifer Mallory in der Nähe des Opfers, das ein mobiles Gerät mit BLE-Chip bestimmter Hersteller verwendet und Bluetooth eingeschaltet hat, so ist dieser Angriff ohne weitere Voraussetzungen durchführbar. Dies kann dazu führen, dass ein empfangendes Gerät zum Opfer wird, und dies kann prinzipiell jedes Gerät sein, das zufällig solch ein Advertisement-Paket erhält. BLE ist eine prominente Bluetooth-Variante, die beispielsweise im Smart-Home-Bereich, aber auch im Medizinbereich schon jetzt sehr verstärkt zum Einsatz kommt.

Dem ‚blutenden‘ Bit (engl. *Bleeding* Bit) wurde offiziell die CVE-Nummer CVE-2018-7080 zugewiesen. Eingeschlichen hat sie sich diese Verwundbarkeit vom Übergang der Version 4.2 zur Version 5.0. Denn der Aufbau des Kopfes für ein Advertising-Paket stellt sich mit der Version 5.0 nun ein klein wenig anders dar, als dies bei Paketen der Version 4.2 der Fall war. Dies kann dazu führen, dass beim Analysieren (engl. *parsen*) solch eingehender Bitfolgen die empfängerseitige Prüfung nicht immer reibungslos bzw. fehlerfrei funktioniert. Erschwerend kommt noch dazu, dass wir es mit zwei Prozessoren zu tun haben, dem *Main Core* und dem *Radio Core.* Der Radio Core analysiert und prüft die eingehenden Bitfolgen, führt also die Rahmenbildung durch, so wie wir dies bereits in Abschn. 3.1 erörtert haben. Jeder auf diese Weise erfolgreich geprüfte, über die Luftschnittstelle eingegangene Rahmen wird anschließend vom Radio Core in eine Warteschlange geschrieben. Der Main Core greift nun auf alle Rahmen zu, die vom Radio Core in der Warteschlange abgelegt sind, prüft diese einzelnen Einträge bei ihrer anschließenden weiteren Bearbeitung jedoch selbst nicht mehr. Um die möglichen Auswirkungen einer derart gestalteten Überprüfung genauer zu verstehen ist, es notwendig, sich zumindest zwei Felder des 16 Bit-Kopfes eines Advertisement-Rahmens genauer anzusehen. Denn für die Bluetooth-Version 4.2 gilt, dass die acht *Most Significant Bits* (MSB) des Kopfes aus sechs Bit für das Längenfeld und zwei weiteren Bit namens *Reserved for Future Use* (RFU) bestehen. In der Version 4.2 hat man also zwei Bit vorgesehen, deren Verwendung noch nicht abschließend geklärt ist und die noch gar nicht genutzt wurden. In der Version 5.0 ‚schluckt‘ das Längenfeld diese zwei Bit, da es nun von sechs Bit auf acht Bit ausgeweitet wurde.

Die zu schildernde Schwachstelle resultiert nun daraus, dass die Arbeitsweisen von Radio Core und von Main Core nur unzulänglich aufeinander abgestimmt worden sind. Denn während der Radio Core seine Prüfung, so wie es die Version 5.0 vorsieht, auf der Basis eines Längenfelds der Größe acht Bit durchführt, holt sich der Main Core die in der Warteschlange abgelegten und vom Radio Core geprüften Rahmen zur weiteren Bearbeitung aus der Warteschlange unter der Annahme, dass für das Längenfeld noch die ursprünglichen sechs Bit aus der alten Version vorgesehen sind. Zur weiteren Verarbeitung der vom Radio Core in die Warteschlange eingetragenen Rahmen verwendet der Main Core nun die Funktion `memcpy()`, welche wir bereits im Abschn. 4.3.1 kennen gelernt.

Diese mit Einführung der Version 5.0 nur unzureichend aufeinander abgestimmten Arbeitsweisen von Radio Core und Main Core können nun bewirken, dass mit dem Empfang eines derartigen Beacons auf der Seite des Opfers weniger Speicherplatz reserviert wird, als für das Ablegen dieses eingehenden Advertisement Beacons tatsächlich erforderlich wäre. Dadurch kommt es zu einem kritischen Speicherüberlauf. Wenn nun hierin Zeiger auf ausführbaren Code enthalten sind, der im ersten Schritt des Angriffs von Mallory über das Versenden vieler Broadcast *Advertisement* beacons bereits in den Speicher des Opfergerätes gelangt sind, denn kann das Opfergerät unter die Kontrolle von Mallory gelangen. So sind alle Voraussetzungen geschaffen, die Mallory benötigt,

um eine Hintertür auf dem verwundbaren BLE-Chip auf dem Gerät des Opfers zu installieren. Über den eingespielten Schadcode können beispielsweise weitere Kommandos entgegengenommen werden, um schlussendlich die vollständige Kontrolle über das Gerät zu erlangen.

Konkret führt Mallory für sein Vorhaben, ein BLE-fähige Gerät in seiner Nähe vollständig zu übernehmen, zwei Teilschritte aus. Im Einzelnen sind dies:

Schritt 1 In einer ersten Phase sendet Mallory eine ganze Reihe von BLE Broadcast *Advertisement* Beacons an das Gerät des Opfers. Die Protokoll-Dateneinheit (engl. *packet data unit,* oder PDU) der Beacons enthält also entweder ein ADV_IND, ADV_DIRECT_IND, ADV_NONCONN_IND oder ein ADV_SCAN_IND. Dieses Verhalten von Mallory ist an sich nicht bösartig. Diese Nachrichten werden empfängerseitig beim Opfer allesamt nach Erhalt in der Warteschlange des BLE-Chips abgelegt. Auch wenn diese Signalnachrichten der eingehenden Beacons für sich genommen nicht schädlich sind, so können diese von Mallory zu einem späteren Zeitpunkt dennoch verwendet werden um Schadcode auszuführen.

Schritt 2 Ist dies geschehen, so kann Mallory mit der zweiten Phase des Angriffs fortfahren: Hierzu sendet Mallory ein weiteres Mal eine herkömmliche BLE-Advertisement-Nachricht, nur, dass Mallory dieses Mal vorab ein einziges Bit innerhalb der PDU ändert. Das Bit ist Bestandteil des Kopfes und gehört zum ursprünglichen zwei Bit großen RFU-Feld der Version 4.2, die ja so in der Version 5.0 gar nicht mehr vorgesehen ist. Es wird dabei vor dem Versenden durch Mallory von 0 auf 1 gesetzt. Diese Änderung bewirkt, dass mit Empfang eines derartigen Beacons auf der Seite des Opfers die Prüfung am Radio Core den Rahmen passieren lässt und anschließend vom Main Core weniger Speicherplatz für diesen reserviert wird, als für das Ablegen dieses eingehenden Advertisement Beacons tatsächlich erforderlich wäre. Es kommt zu einem kritischen Speicherüberlauf. Sind nun hierin Zeiger auf Funktionen enthalten, die auf ausführbaren Code verweisen, der im ersten Schritt des Angriffs über das Versenden vieler Broadcast *Advertisement* Beacons bereits im Speicher enthalten sind, so hat Mallory den Angriff erfolgreich durchgeführt.

Die oben beschriebenen Teilschritte zur Ausnutzung der Verwundbarkeit mit dem Ziel einer entfernten Programmausführung auf der Grundlage der Versendung einer Reihe von *Advertisement* Beacons $ADV(l, m)$ mit dem abschließenden Versenden eines durch Mallory handgeformten Rahmens sind noch einmal in der Abb. 4.18 schematisch dargestellt. Dabei kann der Eintrag im Längenfeld $l = l_1 l_2 \ldots l_8$ den Wert 2^8 nicht überschreiten und die Größe der im Beacon enthaltenen Nutzdaten m ist auf $|m| \leq l \leq 2^8$ beschränkt. Allerdings werden am Main Core nur die Bits $l_1 l_2 \ldots l_6$ als Längenfelder interpretiert, was unter Umständen zu einem Speicherüberlauf führen kann. *WS* bezeichnet die Warteschlange, in die der Radio Core schreibt und aus welcher der Main Core liest.

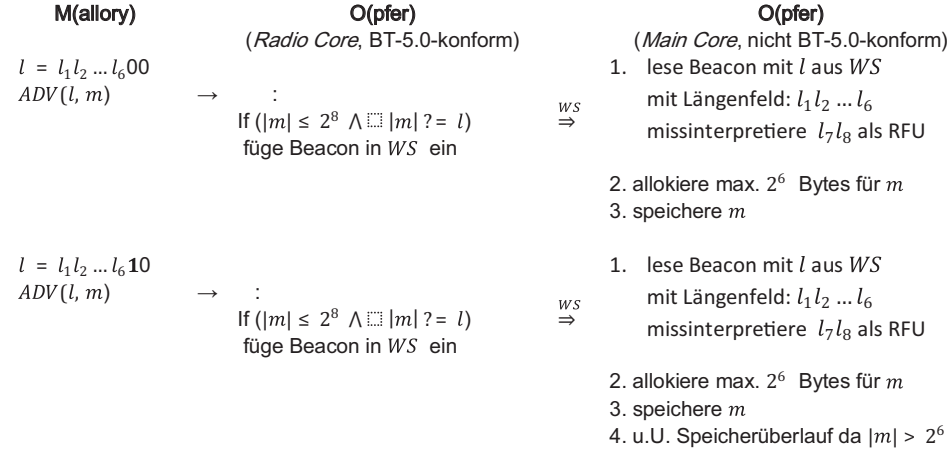

Abb. 4.18 Teilschritte zur Verwundbarkeit mittels eines ‚blutendes' Bits auf ein Gerät mit BLE-Chip. (Quelle: eigene Darstellung)

▶ Die geschilderte Verwundbarkeit auf ausgewählte BLE-Chips (Bluetooth-Version 5.0) macht sich zunutze, dass das Längenfeld eines Advertisement Beacons am Radio Core und am Main Core des Opfers unterschiedlich ausgewertet wird. Während der Radio Core bei der Prüfung eingehender Beacons entsprechend der Bluetooth-5.0-Version (BLE) von einem acht Bit langen Längenfeld ausgeht, wird am Main Core mittels `memcpy()` nur Speicherplatz ausgehend von einem sechs langen Bit Längenfeld allokiert. So kann Mallory einen kritischen Speicherüberlauf auf dem Opfergerät provozieren, indem er ein Advertisement Beacon so manipuliert, dass zumindest eines der in der alten Version 4.2 als RFU vorgesehenen 2 Bit von 0 auf 1 gesetzt wird. Die vorliegende Verwundbarkeit ist **kein Spezifikationsfehler,** sondern, wenn man so will, **eine Implementierungsschwäche, die** für einen Remote-Execution-Angriff durch eine Änderung des Aufbaus eines Advertisement Beacons von BT 4.2 nach BT 5.0 zum Tragen kommen kann.

4.6 Praktische Auswirkungen

Was sind nun tatsächlich die praktischen Auswirkungen der Blueborne-CVE-Familie sowie des hier beschriebenen Angriffs über ungültige Kurven? Einerseits kann man festhalten, dass im Falle von Blueborne das Bereitstellen von Sicherheitsupdates der Firmen Google, Microsoft, Apple und Linux in Kooperation mit Armis vorbildlich funktioniert

hat. Sicherheitsupdates wurden für die entsprechenden Produkte bereitgestellt, noch bevor die einzelnen unter Blueborne zusammengefassten Schwachstellen öffentlich gemacht wurden.

Bedeutet dies nun, dass Blueborne keine praktischen Auswirkungen hat? Dies ist mitnichten der Fall. Denn viele der betroffenen Geräte verfügen noch nicht einmal über die Fähigkeit eines Updates oder aber es lassen, wie im Falle von Android, Patches oftmals sehr lange auf sich warten. Insbesondere ältere Versionen werden teilweise gar nicht mehr mit Updates versorgt. Für den Angriff über ungültige Kurven lässt sich sagen, dass Google und andere Hersteller Patches angekündigt haben. Die Bluetooth SIG kündigte im Spätsommer 2018 an, sich mit der Thematik auseinandersetzen zu wollen.

Ein Grund für die unter Umständen nicht ganz so ausschweifende Nutzung der geschilderten Verwundbarkeiten dürfte in der simplen Tatsache liegen, dass es immer noch viel zu einfach ist über nicht geänderte Defaultpasswörter auf Bluetooth-fähige Geräte zuzugreifen. So ist es für den Angreifer bisher einfach nicht notwendig, die geschilderten Techniken zu verwenden, um an das anvisierte Ziel zugelangen. Eine Reihe von Werkzeugen zur praktischen Ausnutzung der hier beschriebenen Verwundbarkeiten sind verfügbar. Zu nennen sind hierbei insbesondere Proxmark3 Kit, HackRF One, Ubertooth One und Yard Stick One (Yet another radio dongle). Das Proxmark3 Kit ist ein Software Defined Radio (SDR), das auf RFID und NFC abzielt und deren zeitliche Anforderungen beim Senden und Empfangen von Nachrichten erfüllt. HackRF ist ein SDR-Peripheriegerät zum Testen heutiger und zukünftiger Radiotechnologien im Bereich 1 MHz bis 6 GHz. Die Ubertooth-One-Plattform ermöglicht das Übertragen und Empfangen von Daten im 2,4-GHz-Bereich und ist damit auch für Bluetooth-Klasse-1-Geräte geeignet. Das Produkt Yard Stick One zielt auf das Senden und Empfangen von Nachrichten im Bereich kleiner 1 GHz ab. Es sei an dieser Stelle noch einmal ausdrücklich darauf hingewiesen, dass die Verantwortung diese Geräte legal zu nutzen, allein beim Anwender liegt. Details zu der Durchführbarkeit der oben geschilderten Angriffsformen wurden neben anderen auch von Laurent Vetter im Rahmen seiner Masterarbeit [24] erörtert.

4.7 Near Field Communication

4.7.1 Einleitende Bemerkungen

Die *Near Field Communication* (NFC) ermöglicht die kontaktlose Übertragung von Daten, typischerweise im Nahbereich von ca. 10 cm. Als eine Unterart von *Radio Frequency Identification* (RFID) ist NFC kompatibel mit RFID-Standards auf einer Betriebsfrequenz von 13,56 MHz. Geräte wie Smartphones mit integrierter NFC-Fähigkeit können somit mit RFID-Tags und Readern interagieren. NFC unterscheidet hierbei die Betriebsmodi *aktiv* und *passiv*. Als Kommunikationsmodi sind *read/write, card-emulation* sowie *peer-to-peer* spezifiziert.

Ein NFC-fähiges Gerät, das sich im *passiven* Betriebsmodus befindet, initiiert hierbei die Verbindung zu einem Tag oder einem weiteren NFC-fähigen Gerät. Dabei erzeugt das initiierende Gerät ein elektromagnetisches Feld. Es befindet sich im Kommunikationsmodus *read/write* und kann auf das Ziel-Tag lesend oder schreibend zugreifen. Das NFC-fähige Zielgerät hingegen befindet sich im Kommunikationsmodus *card-emulation* und emuliert ein RFID-Tag.

Kommunizieren zwei NFC-fähige Geräte jeweils im aktiven Betriebsmodus, so erzeugen diese wechselseitig elektromagnetische Felder. Der Initiator fungiert dabei als RFID-Reader, das zweite Gerät agiert als RFID-Tag. Nach der Übertragung der Daten wechseln die jeweiligen Rollen beider Geräte, indem der ursprüngliche Initiator sein elektromagnetisches Feld ausschaltet und nunmehr das bisher als RFID-Tag fungierenden Gerät sein elektromagnetisches Feld aktiviert. Beide Geräte befinden sich hierbei im Kommunikationsmodus *peer-to-peer*. Diese Halb-Duplex-Kommunikation ermöglicht erhöhte Reichweiten von bis zu einem Meter.

Das NFC Forum [20] unterscheidet die internationalen Standards NFC-A, NFC-B, NFC-F, NFC-V für höhere Reichweiten sowie P2P. In Anlehnung an diese NFC-Standards existieren verschiedenste standardisierte oder auch nicht standardisierte Tag-Typen für einfachere kostengünstigere Anwendungen bis hin zu teureren Tags für komplexere Anwendungen. Die einzelnen Tag-Typen unterscheiden sich teilweise erheblich, was ihren Speicher, deren Datenrate, sowie deren konfigurierbare Sicherheitsbausteine betrifft. Mehrheitlich verfügen die einzelnen Tag-Typen über ein sogenanntes Antikollisionsverfahren. Denn befinden sich mehrere Tags in der Nähe des Readers, so erkennt das Lesegerät alle Tags gleichzeitig und wäre damit, was den Nachrichtenaustausch angeht, schlichtweg überfordert. Es müssen also Verfahren etabliert sein die festlegen, welches Tag senden darf. Andernfalls würden sich die Signale zu einem nicht mehr dekodierbaren Rauschen überlagern. Antikollisionsverfahren nutzen die Tatsache, dass für gewöhnlich Tags über unterschiedliche IDs verfügen, und detektieren hierüber dasjenige Tag, mit dem ein Nachrichtenaustausch erfolgen soll. So ist beispielsweise das Antikollisionsverfahren der Mifare- Classic-Tags von NXP NFC-A konform und deterministisch für bis zu dreiKaskadierungsstufen von Tag-IDs gestaltet.

Mit dem *NFC Data Exchange Format* (NDEF) kommt ein Nachrichtenformat zum Einsatz, das einen strukturierten Nachrichtenaustausch zwischen NFC-Geräten bzw. NFC-Gerät und Tag ermöglicht. Dabei kann eine NDEF-Nachricht aus verschiedenen sogenannten NDEF-*Records* bestehen, wobei die einzelnen Records jeweils aus einem Record-Kopf und den Nutzdaten des Records zusammengesetzt sind. Auf eine weitere Beschreibung des NDEF wollen wir jedoch an dieser Stelle verzichten.

Neben dem NFC-Reader und dem Tag komplettiert das Backend die NFC-Architektur. Das Backend ist ein eigens zu schützendes Computersystem, das die beim Reader eingehenden Daten entgegennimmt und ablegt. Dabei sollen uns im Rahmen dieses Buches all diejenigen Schwachstellen nicht interessieren, die aus einer zu unbedarften Konfiguration des Backends resultieren.

4.7.2 Authentifizierung zwischen Lesegerät und Tag

Mittels eines Aufgabe-Antwort (engl. *Challenge-Response*) Verfahrens erfolgt die Authentifizierung zwischen NFC-fähigem Lesegerät und Tag. Zur Erinnerung: Bei derartigen Authentifizierungsverfahren stellt eine Partei der anderen Partei eine Aufgabe, die sie nur dann erfolgreich lösen kann, wenn sie im Besitz einer geheimen Information wie beispielsweise eines Schlüssels ist. Mit der Antwort zeigt die Gegenseite der anfragenden Partei, dass sie über die geheime Information verfügt, ohne diese jedoch selbst preiszugeben. Der Vorteil solch einer Vorgehensweise liegt auf der Hand: Durch diese Art der Authentifizierung kann sich eine Partei gegenüber einer anderen ausweisen, ohne dass eine eventuell mithörende dritte Partei Eve in Besitz des Geheimnisses gelangen könnte.

Im Kontext von NFC wird solch ein Challenge-Response-basiertes Protokoll immer dann angestoßen, wenn ein Lesezugriff oder ein Schreibzugriff auf die Speicherstruktur eines Tags erfolgen sollen. Die Abb. 4.19 stellt in vereinfachter Form die Abfolge einer Authentifizierung zwischen Tag und Reader dar, so wie sie bei einigen Mifare-Classic-Modellen von NXP erfolgt. Mit der Nachricht *Auth* fordert der Reader das Tag auf, sich zu authentifizieren. Eigentlich wird hierbei der Befehl *Auth_A* seitens des Readers abgesetzt, um dem Tag mitzuteilen, für welchen Sektor A mit jeweils anderen Schlüsseln innerhalb der Speicherstruktur des Tags er die Authentifizierung anstrebt. Das Tag antwortet mit einer Zufallszahl, einer Nonce n_T. Diese nutzt der Reader zur Berechnung von $a_R = f(n_T)$ wobei die Funktion $f()$ eine 32 Bit-Eingabe in eine 32 Bit-Ausgabe überführt. Im Wesentlichen ist dies eine bitweise Linksverschiebung der 32 Bit der Nonces mit anschließendem Auffüllen des letzten Bits. Der so berechnete Wert a_R wird am Lesegerät verschlüsselt. In der gleichen Nachricht überträgt er ebenfalls seine eigene gewählte Nonce n_R verschlüsselt. Nach erfolgreicher Prüfung dieser Nachricht verfährt das Tag auf ähnliche Weise, indem es seinerseits $a_T = f(n_R)$ berechnet. Hier wird nun allerdings allein der Wert a_T verschlüsselt und an den Reader übertragen. Mit Erhalt des Chiffrates $E(a_T)$ entschlüsselt der Reader das Chiffrat $D(E(a_T))$ und berechnet seinerseits $a_T = f(n_R)$. Stimmen diese beiden Ergebnisse überein, so kann das NFC-Lesegerät schlussfolgern, dass das Tag in Besitz des gemeinsamen Schlüssels ist. Damit gilt die Authentifizierung als erfolgreich durchgeführt.

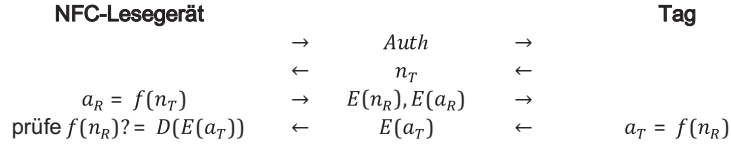

Abb. 4.19 NFC-A-konforme beidseitige Authentifizierung zwischen NFC-Lesegerät und Tag. (Quelle: eigene Darstellung in Anlehnung an [11])

Wurde die erstmalige Authentifizierung für einen Sektor der Speicherstruktur des Tags auf die oben beschriebene Weise erfolgreich durchgeführt, so werden von nun an alle weiteren Nachrichten zwischen NFC-Lesegerät und Tag verschlüsselt übertragen. Dies bedeutet aber auch, dass weitere Authentifizierungen für weitere Sektoren des Tags ebenfalls verschlüsselt erfolgen. Insbesondere wird in diesem Fall nicht mehr die Nonce n_T als Klartext, sondern ebenfalls verschlüsselt als Chiffrat $E(n_T)$ übertragen. Diese Art der Authentifizierung zwischen Lesegerät und Tag bezeichnet man als ineinander geschachtelte (nested) Authentifizierung.

Noch zwei Anmerkungen zu den Notationen: Hinter der Notation $E()$ verbirgt sich die Chiffre CRYPTO-1. Der Aufbau dieser Stromchiffre wurde ursprünglich vom Hersteller geheim gehalten. Allerdings wurde sie mittlerweile einem erfolgreichen *Reverse Engineering* unterzogen, wobei eine ganze Reihe von Schwächen aufgedeckt werden konnten. Eine unrühmliche Rolle spielen dabei die eingesetzten Filterfunktionen, welche aus bestimmten Bits des inneren Zustandes einzelne Ausgabebits für den zu erzeugenden Schlüsselstrom generieren. Denn hier hat es sich herausgestellt, dass die Filterfunktionen zur Erzeugung der einzelnen Ausgabebits des Schlüsselstroms nur eine sehr kleine Untermenge der 48 Bits des jeweiligen inneren Zustandes der CRYPTO-1-Chiffre verwenden. Der auf diese Weise erzeugte Schlüsselstrom wird dann mittels bitweisem exklusivem Oder (XOR) mit den zu verschlüsselnden Klartextbits verknüpft.

Die zweite Anmerkung zur Notation bei der Beschreibung der beidseitigen Authentifizierung betrifft die Nonce. Man mache sich darüber im Klaren sein, dass der Begriff Nonce die Abkürzung für number used only *once* ist. Das Akronym Nonce hat es also in sich, denn es bedeutet schlichtweg, dass hier der Standard bei der tatsächlichen Umsetzung einen Wert fordert, der nicht nur wirklich zufällig erzeugt wurde, sondern darüber hinaus auch tatsächlich nur genau einmal verwendet werden darf.

Nachdem wir nun den Authentifizierungsvorgang zwischen Lesegerät und Tag in seiner einfachen und in seiner verschachtelten Variante erörtert haben, ist es sicherlich sinnvoll, darauf hinzuweisen, dass sogenannte MAGIC-Befehle existieren, mit denen einige Tags dazu veranlasst werden können, die oben geschilderte Authentifizierungsphase vollständig zu überspringen. Darüber hinaus ermöglichen einzelne weitere MAGIC-Befehle das Unterdrücken der Antikollisionsverfahren, andere erlauben es, die ID eines Tags zu ändern.

4.7.3 Ausgewählte Angriffe auf ältere Tags

Auch wenn es mehrheitlich bekannt sein dürfte, dass die Sicherheit von Mifare-Classic-Tags nicht ausreichend ist und der Hersteller in der Zwischenzeit eine Reihe deutlich höherwertigere Tags anbietet, die gegen die nun geschilderten Angriffe immun sind, so ist es dennoch sinnvoll sich einige ausgewählte Angriffe auf diese Tags zu vergegenwärtigen. Schließlich ist es das Ziel dieses Buches bekannt gewordene Schwächen und Verwundbarkeiten auf mobile Systeme zu erörtern und nachvollziehbar zu

machen. Darüber hinaus sind solche veralteten Tags noch häufig im täglichen Gebrauch anzutreffen, beispielsweise bei Bezahlsystemen für den öffentlichen Transport, bei Studentenausweisen, Karten für Fitness-Studios, Golfclub-Mitgliedskarten, kontaktlosen Zugangskontrollsysteme für Bürogebäude und Banken und dergleichen mehr. Im Folgenden sind einige ausgewählte Angriffe auf ältere Tags beschrieben. Neben anderen Autoren und Sicherheitsexperten hat Jan Breig von der Hochschule Offenburg in seiner Abschlussarbeit [4] eine ganze Reihe weiterer bekannter Schwächen älterer NFC-Tags zusammengetragen.

4.7.3.1 Angriffe unter Einbeziehung der Paritätsbits

Zur Erkennung von Bitdrehern bei der Übertragung der Daten zwischen Lesegerät und Tag sieht der NFC-A-Standard nach ISO 14443 A die Verwendung von Paritätsbits vor. Diese Maßnahme zur Erkennung von Fehlern, die sich auf der Übertragungsstrecke eingeschlichen haben, ist üblich und in einer Reihe von Kommunikationsprotokollen vorzufinden, wenn auch oftmals in erweiterter Form. Denn mittels eines einzelnen Paritätsbits ist nur die Erkennung einer ungeraden Anzahl von Bitfehlern möglich. Eine Folge von Bits wird mit einem weiteren Bit angereichert, dem Paritätsbit. Es erhält den Wert ‚1', wenn die Anzahl aller auf ‚1' gesetzten Bits der Folge gerade ist, andernfalls erhält das Paritätsbit den Wert ‚0'. Allerdings hat sich für die im NFC-A vorgeschlagene Ausgestaltung des Protokolls gezeigt, wie wir später sehen werden, dass das Verhalten eines Tags bei der Prüfung dieser Bits von einem Angreifer auf unterschiedliche Art und Weise ausgenutzt werden kann. Doch bevor wir nun auf die daraus resultierenden Verwundbarkeiten eingehen, wollen wir zunächst die Besonderheiten, die der NFC-A-Standard bei der Verwendung von Paritätsbits aufweist, erörtern.

Der Reihe nach: Jedem Klartextbyte $B = b_0b_1b_2b_3b_4b_5b_6b_7$ von zu übertragenden Daten wird ein Paritätsbit p angefügt. Es berechnet sich dabei durch Anwendung von XOR auf die einzelnen Klartextbits sowie durch eine nochmalige Anwendung von XOR auf den so erhaltenen Wert mit dem Wert ‚1', also $p = b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus 1$. Die so entstandene Bitfolge aus Klartextbytes und Paritätsbits $B_1p_1 \ldots B_np_n$ wird als Nächstes gemeinsam verschlüsselt, also $E(B_1p_1 \ldots B_np_n)$, wobei der Standard bei der Verschlüsselung eine recht eigenartige und in seiner Sinnhaftigkeit unverständliche Vorgabe macht: Jedes der Paritätsbits wird mit dem gleichen Bit des Schlüsselstroms verschlüsselt wie das ihm nachfolgende Klartextbit. Dies sei an dieser Stelle erst mal nur als eine Randnotiz erwähnt. Denn die eigentliche für einen Angreifer ausnutzbare Schwäche erwächst aus dem Antwortverhalten des Tags. Das Antwortverhalten eines Tags bei eingehenden Nachrichten eines Readers lässt sich in drei Fälle untergliedern.

Antwortverhalten eines Tags:

Fall1 Ist mindestens eines der Paritätsbits p der entgegengenommenen Nachricht falsch gesetzt, so antwortet das Tag einfach gar nicht.

Fall2 Sind alle Paritätsbits p einer eingehenden Nachricht korrekt gesetzt, aber die dazugehörige Antwort $E(a_R)$ des Lesegerätes erweist sich als nicht passend, so antwortet das Tag mit einem bekannten Fehlercode der Länge vier Bit, den das Tag verschlüsselt versendet $E(Errorcode)$.

Fall3 Sind alle Paritätsbits korrekt gesetzt und die dazugehörige Antwort des Lesegerätes $E(a_R)$ erweist sich als passend so sendet das Tag die verschlüsselte Antwort $E(a_T)$ an das Lesegerät.

Mit diesem Wissen über die Einbettung von Paritätsbits in die Kommunikation nach NFC-A sind wir in der Lage, den nachfolgenden Angriff nachzuvollziehen: Hierbei nutzt der Angreifer das oben geschilderte Verhalten durch Initiieren der NFC-A-konformen Authentifizierung zwischen Lesegerät und Tag mittels Absetzen des *Auth* Befehls (siehe Abb. 4.20). Wie zu erwarten antwortet das Tag hierauf mit dem Absenden seines Nonces n_T. Da die zu erwartende protokollkonforme Antwort des Readers ja verschlüsselt $E(n_R), E(a_R)$ beim Tag eingehen muss, der Angreifer den Schlüssel jedoch nicht kennt, kann er auch nicht die hierfür passenden Paritätsbits berechnen. Mallory sendet in einem ersten Schritt daher eine rein zufällige Bitfolge X anstelle eines ‚passenden' Chiffrats $E(n_R), E(a_R)$, allerdings mit der korrekten Anzahl an Bits, notiert als $|X| = |E(n_R), E(a_R)|$. Da der Klartext-Ausdruck n_R, a_R aus zwei 32 Bit-Werten besteht, also insgesamt acht Byte lang ist und somit vor seiner Verschlüsselung mit acht Paritätsbits angereichert wird, besteht eine Chance von $1/2^8$ für den Angreifer durch das Senden von X, zufällig alle acht Paritätsbits korrekt gesetzt zu haben, so wie sie innerhalb der tatsächlichen Chiffrate $E(n_R), E(a_R) = E(B_1 p_1 \ldots B_4 p_4), E(B_5 p_5 \ldots B_8 p_8)$ gesetzt sein würden. Tritt dieser Fall ein, so antwortet das Tag seinerseits mit einem verschlüsselten vier Bit langen Fehlercode $E(Errorcode)$. Da Mallory die zu Verwendung kommenden Fehlercodes jedoch in der NFC-A-Spezifikation einsehen kann, kann er nun einen *Known-Plaintext*-Angriff starten und im Erfolgsfall hierdurch vier aufeinanderfolgende Bits des Schlüsselstroms $ks := \ldots s_1 s_2 s_3 s_4 \ldots$ in Erfahrung bringen. Wir notieren dies als

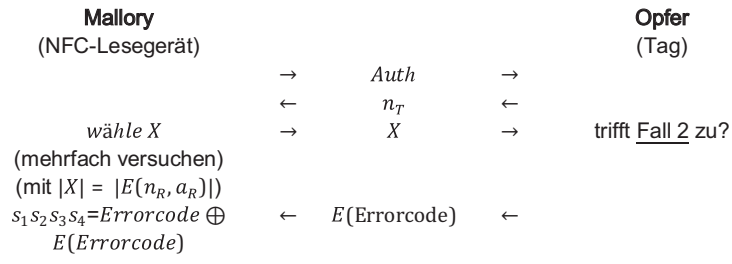

Abb. 4.20 Known-Plaintext-Angriff unter Ausnutzung des Verhaltens am Tag bei Prüfung der Paritätsbits. (Quelle: eigene Darstellung)

$$s_1 s_2 s_3 s_4 = Errorcode \oplus E(Errorcode)$$

Das Wissen um das Auftreten dieser vier Bits im Schlüsselstrom kann Mallory anschließend nutzen, um einen *Brute-Force*-Angriff zur Erlangung des vollständigen Schlüssels durchzuführen, wobei sich durch seine oben geschilderten Vorarbeiten die Komplexität des Brute-Force-Angriffs auf handhabbare 2^{48} reduziert hat. Dieses grundsätzliche Vorgehen zur Vorbereitung eines Known-Plaintext-Angriffs zur Erlangung von vier aufeinanderfolgenden Bits $s_1 s_2 s_3 s_4$ des Schlüsselstroms ist in Abb. 4.20 noch einmal aufgeführt.

4.7.3.2 Der Nested-Authentication-Angriff

Um den unter dem Namen *Nested-Auhentication-Angriff* bekannt gewordenen Angriff nachvollziehen zu können, müssen wir uns vorab ein wenig mit der Speicherstruktur eines Tags auseinandersetzen. So ist der Speicher eines Mifare-Classic-Tags in Sektoren aufgeteilt, die wiederum in Blöcke der Größe 16 Byte unterteilt sind. Ein derartiger Speicheraufbau ist bei allen Mifare-Classic-Tags anzutreffen, auch wenn diese sich teilweise erheblich unterscheiden, was die Anzahl der Sektoren und die darin enthaltenen Blöcke betrifft. Doch diese Details sind für das Gelingen des nachfolgend beschriebenen Angriffs unerheblich. Was bei allen Modellen hingegen übereinstimmt, ist die Tatsache, dass innerhalb des letzten Blocks eines Sektors der Sektorschlüssel sowie die Zugriffsberechtigungen auf den Sektor abgelegt sind.

Mit dem *Nested-Auhentication-Angriff* [8] existiert nun seit bereits gut einer Dekade eine Angriffsform, die es einem Angreifer Mallory ermöglicht, alle übrigen Sektorschlüssel in Erfahrung zu bringen, wenn er einen Sektorschlüssel kennt. Diese Vorbedingung, dass Mallory ein Sektorschlüssel bereits vorliegen muss, klingt erst einmal recht unwahrscheinlich. Wir werden allerdings später sehen, wie Mallory in Besitz eines solchen Schlüssels gelangen kann, wenn dieser nicht beispielsweise durch Verwendung eines Defaultschlüssels auf einfachere Weise in Mallorys Besitz gelangt ist.

Dabei geht unser Angreifer Mallory schrittweise wie folgt vor:

Schritt 1 In der ersten Angriffsphase authentifiziert sich Mallory *zweimal* hintereinander für denjenigen Sektor, dessen Sektorschlüssel ihm bereits vorliegt. Wie gehabt antwortet das Tag bei der erstmaligen Authentifizierung mit der von ihm gewählten Nonce. Wir notieren diese nun mit $n1_T$. Die weitere Authentifizierung erfolgt wie bereits in Abb. 4.20 beschrieben. Sendet Mallory mittels seines Lesegerätes ein zweites Mal einen *Auth*-Befehl, der ein weiteres Mal die Challenge-Response-Authentifizierung für den gleichen Sektor antriggert, so antwortet das Tag nun mit $E(n2_T)$. Diese zweite vom Tag gewählte Nonce wird also verschlüsselt übertragen. Dabei wird $E()$ wie bereits erörtert durch das proprietäre Verschlüsselungsverfahren CRYPTO-1 umgesetzt. Für uns ist an dieser Stelle die Tatsache interessant, dass der Schlüsselstrom $ks0$ dieser Stromchiffre, der zur Verschlüsselung von $E(n2_T)$ verwendet wird, in dieser initialen Phase allein abhängig ist von drei Parametern. Dies sind die ID des Tags, ein Startschlüssel K sowie die Nonce $n1_T$.

Nun ist es ja sodass das Tag seine zu verwendenden Nonces nicht wirklich zufällig wählt, sondern mittels einer Funktion berechnet, die bestenfalls geeignet ist, den Vorgang des zufälligen Wählens vorzugaukeln. Im Falle des im Tag verbauten Pseudo-Zufallszahlengenerators ist es Mallory daher oftmals möglich, mit Bekanntsein von $n1_T$ zu einem Zeitpunkt t_1 auf eine später zum Zeitpunkt t_2 vom Tag verwendete Nonce $n2_T$ zu schließen. Schafft es Mallory also, die Zeitspanne $\mu = t_2 - t_1$ die zwischen einem ersten erfolgreichen Authentifizierungsvorgang und einem zweiten Authentifizierungsvorgang verstreicht, präzise genug einzuschätzen, so gelingt es ihm, die zweite Nonce $n2_T$ des Tags vorherzusagen. Allerdings nehmen wir uns die Freiheit die technischen Feinheiten an dieser Stelle getrost zu überspringen.

Schritt 2 Die zweite Phase des *Nested-Auhentication-Angriffs* besteht für Mallory nun darin, sichein weiteres Mal für denjenigen Sektor der Speicherstruktur zu authentifizieren, für den ihm der Sektorschlüssel bereits vorliegt. Über diesen Vorgang erhält Mallory also die als Klartext gesendete Nonce $n1_T$. Nun erfolgt eine zweite Authentifizierung, diesmal jedoch für einen Sektor derjenigen Speicherstruktur des Tags, für den Mallory eben noch nicht im Besitz des dazugehörenden Sektorschlüssel ist. Hat Mallory in Schritt 1 die Zeitspanne $\mu = t_2 - t_1$ zwischen den zwei Authentifizierungsvorgängen, in denen das Tag die Nonces $n1_T$ und $n2_T$ verwendet, präzise genug gemessen, so hilft ihm dies nun. Denn die darauffolgende zweite Authentifizierung für den Sektor, für den Mallory keinen Sektorschlüssel besitzt, muss in exakt dem gleichen zeitlichen Abstand μ erfolgen wie die zeitliche Distanz zwischen den zwei Authentifizierungen aus Schritt 1 für einen Zugang auf einem Sektor mit schon bekanntem Sektorschlüssel. Gelingt Mallory dies, so kann er davon ausgehen, dass in dem von ihm empfangenen Chiffrat die gleiche Nonce $n2_T$ vom Tag gewählt wurde und verschlüsselt wurde. Mehr noch: Mallory weiß auch, dass in dieser initialen Phase im Schlüsselstrom zur Verschlüsselung von $E(n2_T)$ wiederum $ks0$ eingegangen ist. Nun ist er in der Lage, durch Ausführen eines bitweisen XOR der zweiten Nonce auf das empfangene Chiffrat, also $n2_T \oplus E(n2_T)$, Teile des Schlüsselstroms zu berechnen. Konkret gelingt es Mallory auf diese Weise, 32 Bit des Schlüsselstroms in Erfahrung zu bringen, also eine Bitfolge $s_1 s_2 \cdots s_{32}$. Denn auch wenn $|E(n2_T)|$ aus 36 Bit besteht, da auch bei $E(n2_T) = E(B_1 p_1 \ldots B_4 p_4)$ die vier Paritätsbits mit den gleichen Schlüsselstrombits verschlüsselt werden wie vier der Klartextbits von $n2_T$, so erhält Mallory auf diese Weise insgesamt 32 und nicht 36 Bit des Schlüsselstroms. Die Autoren stellen in [8] heraus, dass bei präzisem Timing der Angreifer tatsächlich nur zwischen zwei bis vier Versuchen benötigt bis er auf die dargestellte Weise erfolgreich den verwendeten Schlüsselstrom rekonstruieren kann.

4.7.3.3 Der Darkside-Angriff

Der im vorherigen Abschnitt geschilderte *Nested-Auhentication-Angriff* kann nur dann erfolgversprechend ausgeführt werden, wenn Mallory zumindest über einen der verwendeten Sektorschlüssel verfügt. Der nun geschilderte *Darkside-Angriff* [4] wurde bereits im Jahre 2009 von dem Sicherheitsexperten Courties in einem technischen Report

festgehalten. Zumindest die Grundidee dieses Angriffs wollen wir an dieser Stelle skizzieren. Für weiterführende Details sei auf den technischen Report verwiesen.

Auch der Darkside-Angriff nutzt wiederum das bereits geschilderte Protokollverhalten bei der Prüfung der Paritätsbits aus, um Teile des Schlüsselstroms rekonstruieren zu können. Darüber hinaus nutzt er die Tatsache aus, dass aufgrund der sehr schlechten statistischen Eigenschaften der CRYPTO-1-Chiffre in 75 % aller Fälle das Chiffrat auf der Nonce $E(n_R)$ mit dem gleichen Schlüsselstrom verschlüsselt wurde. Dies hat dann für die Bits des nachfolgend erzeugten Schlüsselstroms die verheerende Auswirkung, dass dieser ab dann nur noch von drei inneren Zustandsbits der CRYPTO-1-Stromchiffre abhängt.

Doch der Reihe nach: Beim Darkside-Angriff initiiert Mallory viele Authentifizierungsvorgänge zwischen dem Lesegerät und dem Tag. Dabei ist Mallory bestrebt, mittels präziser zeitlicher Initiierung der einzelnen Authentifizierungsvorgänge die vom Tag gewählte Nonce ni_T konstant und vorhersagbar zu halten. Mallory antwortet jeweils mit einer ‚zufälligen' Bitfolge *X,* wobei auch hier wie schon bei den vorherigen Angriffen auf der Grundlage von Paritätsbits gelten muss, dass $|X| = |E(n_R), E(a_R)|$. Zur Erinnerung: Das Antwort-Chiffrat des Readers enthält nach jedem Byte jeweils ein Paritätsbit dergestalt, dass $E(n_R), E(a_R) = E(B_1 p_1 \ldots B_4 p_4), E(B_5 p_5 \ldots B_8 p_8)$. Eigentlich ist die Bitfolge X bei diesem Angriff von Mallory doch nicht ganz zufällig gewählt. Tatsächlich ändert Mallory bei jedem weiteren Authentifizierungsvorgang einzelne Paritätsbits. Konkret ändert er in der Bitfolge $X = x_1 \ldots x_{72}$ die Bits x_9, x_{18}, x_{27} bis x_{72}. Und zwar immer dann, wenn Mallory einen Fehlercode $E(Errorcode)$ als Antwort vom Tag erhält, so speichert er das auf diese Weise gewonnene Chiffrat.

Nun verfeinert Mallory sein Verhalten noch einmal indem er weitere Authentifizierungsvorgänge mit dem Tag initiiert. Diesmal jedoch werden neben einzelnen Paritätsbits auch weitere, dedizierte Bits der Bitfolge $X = x_1 \ldots x_{72}$ geändert. Mallory ändert jeweils die letzten drei Bit des vierten Bytes B_4, wobei er die Bytes B_1 bis B_3 und B_5 bis B_8 nicht verändert und konstant hält. Darüber hinaus variiert er für diese sieben weiteren Authentifizierungsversuche nur die Paritätsbits p_4 bis p_8, nicht jedoch die Paritätsbits p_1 bis p_3. In der gesamten Bitfolge X ändert Mallory also demnach die Bits $x_{33}, x_{34}, x_{35}, x_{36}$, sowie die Bits x_{45}, x_{54}, x_{63} und x_{72}. Reagiert das Tag auf derartige eingehende Nachrichten mit einem verschlüsselten Fehlercode $E(Errorcode)$ als Antwort, so speichert Mallory auch diese Antworten des Tags.

An dieser Stelle kommen die oben bereits erwähnten drei inneren Zustandsbits der CRYPTO-1-Chiffre zum Tragen: Denn mit einer Abhängigkeit von nur drei Bits zeigt es sich, dass ein weiterer Abschnitt, nennen wir ihn *ks,* des gesamten nachfolgend generierten Schlüsselstroms von jeweils nur noch zwei weiteren inneren Zuständen des CRYPTO-1-Chiffre Bausteins mit jeweils der Länge von 21 Bit abhängen. Diese Tatsache ist dem inneren Aufbau des Verschlüsselungsverfahrens und insbesondere der Verwendung der Filterfunktionen geschuldet, auf welche wir jedoch hier nicht im Detail eingehen wollen. Mit dem Wissen um dieses Verhalten bzw. dieser dem CRYPTO-1-Baustein innewohnenden Sicherheitsschwäche ist Mallory nun allerdings in der Lage,

die jeweils 2^{21} möglichen inneren Zustände, die für die Erzeugung des Schlüsselstromabschnitt *ks* verantwortlich sind, in jeweils zwei Tabellen T_1 und T_2 abzulegen.

Nun hat Mallory alle Vorarbeiten durchgeführt, sodass er sich als Nächstes wiederum den gespeicherten Fehlercodes $E(Errorcode)$ zuwenden kann. Für jeden der gespeicherten Fehlercodes berechnet Mallory den verwendeten Teilschlüsselstrom $s_1s_2s_3s_4 = E(Errorcode) \oplus Errorcode$, indem er den Fehlercode und das jeweilige gespeicherte Chiffrat mit XOR verknüpft.

Daraufhin setzt Mallory jeden Eintrag t_i mit $1 \leq \mathrm{i} \leq 2^{21}$ aus der Tabelle T_1 und jeden Eintrag t_j mit $1 \leq \mathrm{j} \leq 2^{21}$ der Tabelle T_2 als innere Zustände für die relevanten zwei Mal 21 Bits der Chiffre ein und prüft ob die daraus resultierende Bitfolge einen Teilabschnitt $s_1s_2s_3s_4$ enthält. Wir notieren diese Überprüfung als $\ldots s_1s_2s_3s_4 \ldots ? = ks = \mathrm{CRYPTO\text{-}1}(t_i, t_j)$ für alle $1 \leq \mathrm{i} \leq 2^{21}$ und alle $1 \leq \mathrm{j} \leq 2^{21}$. Man mache sich klar, dass die gewählte Notation streng genommen ein wenig irreführend ist, da t_i und t_j dem Baustein CRYPTO-1 ja nicht als Eingangsparameter übergeben werden, sondern die jeweilige innere Bitbelegung nach dem eigentlichen Start darstellen.

Mit jeder derartigen Prüfung über alle Einträge t_i und t_j in den Tabellen und der gespeicherten verschlüsselten Fehlercodes wird die Menge möglicher relevanter zwei 21-Bit Kandidaten für den inneren Zustand des CRYPTO-1 Bausteins eingeschränkt, bis schließlich bei praktischer Durchführung dieses Angriffs in durchschnittlich 15 Sekunden der Sektor-Schlüssel in Erfahrung gebracht werden kann. Dies gilt allerdings nur für 3/4 aller Angriffe. In allen restlichen Fällen war der so erzeugte Schlüsselstrom nicht nur von den hier betrachteten zwei Mal 21-Bit des inneren Zustandes abhängig, sondern noch von weiteren, bei dieser Angriffsform nicht betrachteten inneren Zustands-Bits des CRYPTO-1 Bausteins.

▶ Die geschilderten Verwundbarkeiten von Mifare Classic Tags, die der *Nested Authentication* Angriff sowie der *Darkside* Angriff ausnutzen, beruhen auf dreierlei Aspekten. Zum einen i) einer unvorteilhaften Verwendung von *Paritätsbits* und zum zweiten ii) der *Reproduzierbarkeit von Noncen* bei einer präzisen zeitlichen Wiederholung des Authentifizierungsvorganges und zum dritten iii) der Tatsache, dass einzelne durch den Baustein CRYPTO-1 erzeugte Schlüsselstrombits nur von der Belegung einiger weniger innerer Zustandsbits des Crypto-Bausteines abhängig sind. Dies sind **Spezifikationsfehler** von NFC-A,der Stromchiffre sowie des verwendeten PRNG.

4.7.3.4 Der Hardnested Angriff

Im Jahre 2015 haben Carlo Meijer und Roel Verdult [18] einen Angriff vorgestellt, der nicht nur bei Mifare-Classic-Tags zielführend ist, sondern auch bei seinen gehärteten Nachfolgern wie den EV1-Versionen erfolgreich angewandt werden kann. Dieser unter dem Namen *Hardnested Attack* bekannt gewordene Angriff setzt im Wesentlichen bei den schon in den vorangegangenen Abschnitten angeklungenen Schwächen an:

1. Den im Chiffrat der Nonce $E(n_T) = E(B_1p_1 \ldots B_4p_4)$ eingewebten Paritätsbits p_i.
2. Der Tatsache, dass die verschlüsselten Paritätsbits $E(p_i)$ und das ihnen jeweilige direkt nachfolgende erste Bit $E(b_{i+1,1})$ aus dem Folgebyte B_{i+1} mit dem gleichen Schlüsselstrombit verschlüsselt werden.

Allerdings, so werden wir gleich erfahren, haben die Sicherheitsforscher Meijer und Verdult es geschafft, lineare Abhängigkeiten zwischen dem Chiffrat, den Paritätsbits und einzelnen Bits des Schlüsselstroms aufzudecken. Sie haben gezeigt, wie sich derartige Abhängigkeiten für einen Angriff auch auf die bereits gehärteten Tags verwenden lassen. Dabei ist es Mallorys Ziel die Menge aller denkbaren Schlüssel ausgehend von diesen beiden Beobachtungen schrittweise einzuschränken, bis schließlich die Menge der noch übrig gebliebenen Kandidaten für den Schlüssel so geschrumpft ist, dass der tatsächlich verwendete Schlüssel dann idealerweise durch einen *(offline)* Brute-Force-Angriff auf einem zu diesem Zeitpunkt moderaten Suchraum in Erfahrung gebracht werden kann. Allerdings weisen die Erfinder des *Hardnested*-Angriffs auch darauf hin, dass zwar der Angriff mit hoher Wahrscheinlichkeit, jedoch nicht mit absoluter Sicherheit erfolgreich durchgeführt werden kann. Dies verwundert nicht, beruht der Angriff doch auf der Technik der *differentiellen Kryptoanalyse,* bei der Mallory Wahrscheinlichkeiten möglicher Schlüssel abschätzt, um daraus dann die wahrscheinlichsten Schlüssel zu ermitteln. Der wesentliche Gedanke, welcher der differentiellen Kryptoanalyse zugrunde liegt, nämlich die Auswirkung von Differenzen in Klartextpaaren auf die Differenzen der daraus resultierenden Geheimtextpaare in die Analyse einzubeziehen, erweist sich vor den oben genannten beiden Schwächen dieses kryptografischen Verfahrens als besonders scharfes Schwert.

Doch der Reihe nach: Mallory initiiert für den Angriff eine massive Anzahl geschachtelter *(nested)* Authentifizierungsvorgänge mit dem Tag. Für diese Angriffsform ist es erforderlich, geschachtelte Authentifizierungsvorgänge anzustoßen, da für Mallory nicht der eigentliche Klartext in Form der Nonce n_T von Interesse ist, sondern ihr Chiffrat $E(n_T)$.

Mallory speichert also alle Antworten des Tags der Form $E(n_T) = c_1c_2 \ldots c_{72}$, wobei die Sicherheitsexperten für einen erfolgreichen Angriff durchschnittlich 10.000 verschlüsselte Nonces prognostizieren, um in 10–20 min [18] auf den Schlüssel zu schließen.

Neben einer ganzen Reihe von weiteren Aspekten, die in den Hardnested-Angriff einfließen, um einzelne Schlüssel aus dem Möglichkeitsraum auszuschließen, lässt Mallory beispielsweise auch die folgende Berechnung eingehen: Gelingt es ihm, auf die oben geschilderte Weise alle Chiffrate mit allen möglichen Varianten für das erste Byte $c_1c_2 \ldots c_8$ zu erhalten, so führt Mallory die folgenden Berechnungen aus, die auf den XOR-Summeneigenschaften beruhen und die Carlo Meijer im Jahre 2015 im Rahmen seiner Masterarbeit für das Zusammenspiel von Nonce, Paritätsbit und der korrespondierenden Folge an Schlüsselbits herausgearbeitet hat [17]. Hierbei betrachtet Meijer das i-te verschlüsselte Byte $B_i = c_{i,1}c_{i,2} \ldots c_{i,8}$ der Tag-Nonce sowie das jeweilige hierzu

korrespondierende Paritätsbit $c_{i,9} = E(p_i)$ und präsentiert einen Ausdruck für die bei der Verschlüsselung eingehenden Folge an Schlüsselstrombits $s_{i,1}s_{i,2} \ldots s_{i,8}$. Dieser Ausdruck lautet $s_{i1} \ldots \oplus s_{i8} \oplus 1 = c_{i1} \ldots \oplus c_{i8} \oplus c_{i,9}$. Man vergegenwärtige sich, dass dieser Ausdruck insbesondere unabhängig ist von dem eigentlichen Klartext, also dem vom Tag gewählten Nonce n_T.

In seiner Masterarbeit leitet Meijer anschließend einen Ausdruck her, den er Summeneigenschaft nennt. Die Summeneigenschaft ergibt für gewöhnlich eine Ganzzahl, die aus dem Wertebereich 0–256 stammt. Interessant dabei ist, dass dieser Wert eine Eigenschaft des internen Zustandes der Chiffre zu einem bestimmten, hier nicht weiter präzisierten Zeitpunkts ist. Dieser Summenwert lässt sich auf zweierlei Arten berechnen. Zum einen lässt er sich für einzelne Bytes B_i mit $1 \leq \mathrm{i} \leq 4$ der verschlüsselten Nonce $E(n_T) = E(B_1p_1 \ldots B_4p_4)$ berechnen, gemeinsam mit deren korrespondierenden verschlüsselten Paritätsbits. In einer an dieser Stelle sehr vereinfacht wiedergegebenen Darstellung definiert Meijer die Summe S des i -ten verschlüsselten Bytes $B_i = c_{i,1}c_{i,2} \ldots c_{i,8}$ mit dem verschlüsselten Paritätsbit $c_{i,9} = E(p_i)$ als Ausdruck $S := \sum_{0 \leq i \leq 255} c_{i1} \ldots \oplus c_{i8} \oplus c_{i,9}$. Hierbei geht das Byte B_i in all seinen 256 möglichen Ausprägungen mit den jeweiligen korrespondierenden Paritätswert $c_{i,9}$ ein, während die restlichen Bytes der Nonce konstant gelassen werden. Interessant ist nun, dass der so ‚konstruierte' bzw. definierte Summenwert auch auf eine zweite Art berechnet werden kann. Nämlich über die Verrechnung interner Zustandsbits z der Filterfunktionen zum besagten Zeitpunkt so wie sie in der CRYPTO-1-Stromchiffre enthalten sind. Es gilt, an dieser Stelle wiederum nur in sehr vereinfachter Form dargestellt, dass $\mathrm{S} = \sum_{0 \leq \mathrm{i} \leq 255} \mathrm{Filterfunktion}\left(\mathrm{z_{i,1}z_{i,2}} \ldots\right)$. Meijer ist es durch diese zwei Arten der Berechnung der Summe S also gelungen, eine Beziehung zwischen dem intern anliegenden Zustand der Chiffre, und dem aktuell erzeugten Chiffrat herzustellen.

Man muss sich bewusst machen, dass über das durch Meijer herausgearbeitete Kriterium der Summeneigenschaft der einzelnen verschlüsselten Bytes B_i einer Nonce des Tags die Anzahl der möglichen gültigen Schlüsselkandidaten signifikant reduziert werden kann, indem sie für alle möglichen 256 Ausprägungen des ersten verschlüsselte Bytes eines Nonce zur Anwendung gelangt. Die obigen Ausdrücke werden also in einem ersten Schritt für B_1 durchgeführt. Als mögliche Schlüsselkandidaten bleiben nur diejenigen Bitfolgen übrig, deren Summeneigenschaft für B_1 einen Wert zwischen 0 und 256 ergeben. Alle anderen bis zu diesem Zeitpunkt noch denkbaren Schlüsselkandidaten werden aussortiert. In [18] haben Meijer und Verdult diese Grundidee dann noch weiter verfeinert, um auf diese Weise die Menge der möglichen gültigen Kandidaten für Schlüssel noch deutlicher zu reduzieren. Für weitere Techniken wie beispielsweise das Aufteilen der Summeneigenschaften in gerade Summen und ungerade Summen, und die Details der differentiellen Kryptoanalyse, so wie sie in den Hardnested-Angriff Eingang findet sei der Leser auf [17, 18] verwiesen.

▶ Die geschilderten Verwundbarkeiten von Mifare-Classic-Tags und den gehärteten Nachfolgemodellen, auf denen der Hardnested-Angriff basiert,

macht sich die *Summeneigenschaft* zunutze, mit der ein Angreifer über die unvorteilhafte Verwendung der Paritätsbits einen Zusammenhang (Linearität) zwischen dem aktuellen Chiffrat einer verschlüsselten Nonce vom Tag und dem inneren Zustand der Chiffre herstellen und für einen Brute-Force-Angriff auf einer reduzierten Menge von Schlüsselkandidaten ausnutzen kann. Der Angriff ist ein **Spezifikationsfehler** der verwendeten Stromchiffre sowie der Verwendung der Paritätsbits.

4.7.4 Erhöhte Sicherheit mit neueren Tag-Modellen

Mit einer nachfolgenden Generation an Tags, wie den *Mifare-Plus*-Tag-Modellen wurde eine Reihe zusätzlicher sicherheitsrelevanter Erweiterungen vorgenommen. Hier lohnt es sich, genauer auf die einzelnen Modelle zu schauen, bevor eine Auswahl für den operativen Betrieb erfolgt. Die *Mifare-Plus*-Tag sind in vier verschiedenen Sicherheitsstufen konfigurierbar, wobei die Stufe 0 keinerlei Sicherheit bietet und allein zum Aufspielen der Schlüssel für die höheren Sicherheitsstufen zu verwenden ist. In der Stufe 0 sollte ein Tag daher keinesfalls im operativen Modus genutzt werden. In der höchsten Sicherheitsstufe 3 wird dann ausschließlich die Blockchiffre AES eingesetzt, während in den Stufen 1 und 2 immer noch das veraltete und – wie in Abschn. 4.7.3 beschrieben – als unsicher bewiesene Verfahren CRYPTO-1 zum Einsatz kommt. Zur Verschlüsselung und zur Integritätsprüfung werden unterschiedliche Schlüssel eingesetzt. Allerdings muss nicht jede gesendete Nachricht zwingend mit einem *Message Authentication Code* (MAC) unter Verwendung der Blockchiffre AES gesichert sein. Einige *Mifare-Plus*-Tag-Modelle erlauben es, dass für reine Lese-Befehle die MAC-Prüfung ausgeschaltet werden kann und einzig Schreibbefehle zwingend einer MAC-Prüfung unterliegen. In diesen Fällen ist es ratsam, über das jeweilige *Access-Byte* eines jeden Sektors der Speicherstruktur sorgfältig festzulegen, welche Bereiche aus der Speicherstruktur des Tags auch dann gelesen werden dürfen, wenn keinerlei MAC-Prüfung bei einem eingehenden Lese-Befehl erfolgt.

Diese neuere Generation von Tags verfügt darüber hinaus noch über eine Reihe weiterer Sicherheitsmechanismen. Einige davon wollen wir an dieser Stelle kurz vorstellen: Über einen *Proximity-Check* soll verhindert werden, dass die Authentifizierung über einen Relay-Knoten weitergeleitet wird, sodass die Nachrichten n_T und $E(n_R), E(a_T)$ beispielsweise unter Zuhilfenahme eines leistungsstärkeren Gerätes erstellt wurden. Technisch kann solch ein Proximity-Check umgesetzt werden, indem mittels der Paketumlaufzeit (engl. *Round-Trip-Time* [RTT]) die Dauer der einzelnen Schritte eines Authentifizierungsvorganges gemessen werden und mit einer oberen hierfür vorab vorgesehenen zeitlichen Schranke Δ abgeglichen werden. Wir erinnern uns: Allgemein verstehen wir unter der Paketumlaufzeit die für den Hin- und Rücktransport eines Paketes bzw. Rahmens über das Netzwerk benötigte Zeit. Da bei der NFC-Technologie die Kommunikation zwischen Tag und Lesegerät jedoch typischerweise eine sehr kurze

Einfach-Hop-Distanz darstellt werden Umlaufzeiten hier ausgesprochen gering sein und insbesondere keinen größeren zeitlichen Schwankungen unterworfen sein. Damit ist die Einbeziehung der RTT im Kontext von NFC ideal um zeitliche Ungereimtheiten während des Authentifizierungsvorganges zwischen Tag und NFC-Lesegerät sichtbar zu machen. Denn diese werden zwangsläufig messbar sein, sollte ein Angreifer Mallory einen Relay-Knoten zwischen die Kommunikation von Lesegerät und Tag platziert haben. Eine mögliche Ausgestaltung solch einer Authentifizierung mit Proximity-Check ist in der Abb. 4.21 aufgeführt. Das Lesegerät misst mit seiner lokalen Uhr die Zeitpunkte t_0 und t_1 und erhält durch $t_1 - t_0$ die Zeitspanne, die zwischen Versenden der ersten Nachricht und Erhalten der letzten Nachricht vergeht. Wenn nun also der Fall $t_1 - t_0 \geq \Delta$ eintritt, wird die Authentifizierung abgebrochen und gilt als nicht erfolgreich durchgeführt. Oder umgekehrt: Die in Abb. 4.21 dargestellte Authentifizierung ist nur dann erfolgreich, wenn wie bereits erörtert $f(n_R) = D(E(a_T))$ gilt und wenn darüber hinaus gleichzeitig gilt, dass $t_1 - t_0 < \Delta$.

Bevor wir uns den weiteren Sicherheitsmechanismen zuwenden, hier noch zwei Anmerkungen zu obigem Verfahren: Sicherlich werden Sie bemerkt haben, dass über $t_1 - t_0$ eigentlich zweimal die RRT eingeht, da hiermit ein Zeitraum für den Hin- und Rücktransport von vier Rahmen/Paketen beschrieben ist. Und weiter: Da die Zeiten t_0 und t_1 beide von der gleichen lokalen Uhr des Lesegerätes stammen, birgt dieses Verfahren keinerlei Problematik, was eine eventuelle Ungenauigkeit, hervorgerufen durch Uhren-Drift, anbelangt. Mehr noch: Die lokale Uhrzeit muss nicht einmal die ‚echte' Uhrzeit darstellen solange sie für den Schwellenwert Δ geeignet justiert ist.

Ein *Transaktions-MAC* ermöglicht eine Prüfung eingehender Nachrichten mittels *Message Authentication Code* am Backend, also dem Computersystem, welches hinter dem eigentlichen Lesegerät liegt. Hierfür ist ein weiterer Schlüssel notwendig, der am Tag und am Backend vorliegen muss. Daneben ermöglichen *Originalitätsprüfungen* die Überprüfung, ob ein Tag vom eigenen Hersteller stammt. Je nach vorliegendem Tag Modell geschieht dies auf unterschiedliche Art und Weise: Einige der Tag-Modelle verwenden einen symmetrischen Originalitätsschlüssel zur Ausführung einer Authentifizierung (siehe Abb. 4.21) unter Verwendung von AES für die Verschlüsselungsfunktion

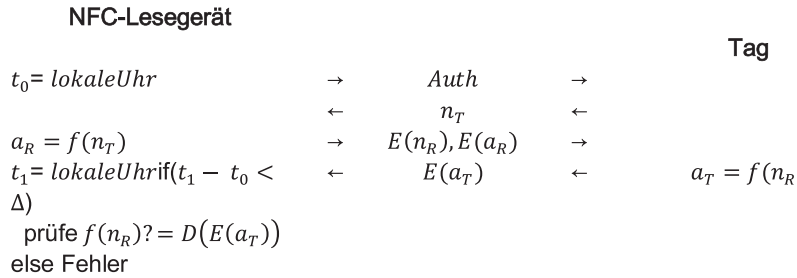

Abb. 4.21 Beidseitige Authentifizierung zwischen NFC-Lesegerät und Tag unter Einbeziehung eines Proximity-Checks. (Quelle: eigene Darstellung)

$E()$. Leistungsstärkere Tag-Modelle sind in der Lage, zur Originalitätsprüfung asymmetrische Krypto-Komponenten auf Basis von elliptischen Kurven zu verwenden. Letzteres gilt für das Modell Mifare Plus EV1.

4.7.5 Authentifizierung bei Mifare DESFire EV1

Mifare DESFire stammt in seiner ersten Version aus dem Jahre 2002. Daher wundert es auch nicht in den frühen Versionen noch DES bzw. Triple-DES als Verschlüsselungsverfahren vorzufinden. Dieses wurde allerdings recht schnell um die Blockchiffre AES ergänzt, sodass nunmehr drei Verschlüsselungsverfahren zur Auswahl auf derartigen Tags existieren. An dieser Stelle beschreiben wir nur den noch als sicher geltenden Authentifizierungsvorgang unter Verwendung von AES. Der ursprüngliche Authentifizierungsvorgang basierend auf DES bzw. Triple-DES, ist ein wenig anders aufgebaut, beruht aber ebenfalls auf einem Challenge-Response-Ansatz. Da hierzu jedoch ein Seitenkanalangriff aus dem Jahre 2010 bekannt ist, begnügen wir uns damit, die zum Zeitpunkt des Verfassens dieser Zeilen als noch sicher geltende Variante vorzustellen.

Mit dem Empfang eines *Auth*-Befehls wählt das Tag eine 128 Bit-Nonce n_T und sendet daraufhin das nun mit AES verschlüsselte Chiffrat $E(n_T)$ an den Reader. Der Reader entschlüsselt das Chiffrat und erhält dadurch $n_T = D(E(n_T))$. Nun bearbeitet er n_T, indem er eine bitweise Linksverschiebung um acht Positionen vornimmt: $ln_T = RotLeft(n_T)$. Anschließend wählt der Reader seinerseits einen 128 Bit-Nonce n_R. Diese geht in die Berechnung zweier Chiffren c_1 und c_2 ein. Den ersten Wert erhält er durch bitweises Anwenden von XOR auf seiner Nonce n_R und der entgegengenommenen verschlüsselten Nonce n_T, also $c_1 = E(n_R \oplus E(n_T))$. Der zweite Wert ergibt sich durch bitweises XOR vom ersten Wert mit der um acht Positionen geshifteten Nonce des Tags und anschließender Verschlüsselung, ergo $c_2 = E(ln_T \oplus c_1)$.

Mit dem Empfang beider Chiffrate entschlüsselt das Tag diese, um anschließend seinerseits einige XOR-Operationen sowie Rechts- und Linksrotationen auf den so gewonnenen Klartexten durchzuführen. War die Prüfung $n_T? = RotRight(ln_T)$ erfolgreich, so erstellt das Tag seinerseits ein Chiffrat $E(ln_R \oplus c_2)$ um dieses an das Lesegerät zu übertragen. Mit der Entschlüsselung dieser Chiffre, dem Erhalt von ln_R und einer abschließenden Prüfung nach einer Rechtsrotation um acht Positionen endet ein erfolgreicher Authentifizierungsvorgang (Abb. 4.22).

4.7.6 Aus der Praxis

Mit Proxmark3, vorzugsweise in den Versionen EVO oder RDV4, lassen sich Kommunikationen zwischen einem Tag und einem Lesegerät abhören. Es ist das Pentest-Werkzeug für NFC- und für RFID-Technik. Darüber hinaus werden Befehle angeboten, um Magic Tags zu lesen und zu schreiben, ohne Schlüsselmaterial zu verwenden. Leser, die nach

NFC-Lesegerät				Tag
	→	$Auth$	→	
	←	$E(n_T)$	←	wähle n_T
$n_T = D(E(n_T))$				
$ln_T = RotLeft(n_T)$ wähle n_R $c_1 = E(n_R \oplus E(n_T))$ $c_2 = E(ln_T \oplus c_1)$				
	→	c_1, c_2	→	$n_R \oplus E(n_T) = D(c_1)$ $ln_T \oplus c_1 = D(c_2)$ $ln_T = ln_T \oplus c_1 \oplus c_1$ $n_T ? = RotRight(ln_T)$ $n_R = n_R \oplus E(n_T) \oplus E(n_T)$ $ln_R = RotLeft(n_R)$ $E(ln_R \oplus c_2)$
$ln_R \oplus c_2 = D(ln_R \oplus c_2)$ $ln_R = ln_R \oplus c_2 \oplus c_2$ $n_R ? = RotRight(ln_R)$	←	$E(ln_R \oplus c_2)$	←	

Abb. 4.22 Authentifizierung mit AES beim Tag-Modell Mifare DESFire EV1. (Quelle: eigene Darstellung in Anlehnung an [14])

dem Studium der in den vorangegangenen Abschnitten beschriebenen Angriffsformen Nested-Authentication-Angriff, Darkside-Angriff, sowie Hardnested-Angriff davon ausgehen, dass diese Angriffe zu komplex und daher nur von sehr wenigen Experten zu bewerkstelligen sind, dürfen sich nicht in Sicherheit wiegen. Denn mit Proxmark 3 lassen sich auch die beschriebenen Angriffe Nested-Authentication Angriff, Darkside-Angriff, sowie Hardnested-Angriff durchführen. Derartige Angriffe lassen sich relativ komfortabel realisieren, wenn es gelingt, das Gerät in der Nähe eines NFC-Tags zu platzieren. Mit einem Raspberry Pi als Backend, gespeist über eine Powerbank, passt die benötigte Architektur in jeden Rucksack und ist damit von einem Angreifer der Kategorie Skriptkiddie durchführbar. Darüber hinaus lassen sich auf Github Implementierungen finden, mit denen der Darkside-Angriff auch unter Verwendung gewöhnlicher Reader möglich ist.

4.8 Zusammenfassung

Wir tragen die erörterten Verwundbarkeiten auf die in diesem Kapitel behandelten Personal Area Networks auf Basis der Technologien Bluetooth und NFC noch einmal kompakt zusammen.

Nachdem wir uns zunächst mit den unter Bluetooth vorgesehenen Pairing-Modi und Pairing-Phasen vertraut gemacht haben, haben wir uns im Weiteren angeschaut, was es mit dem Einsatz von LE Privacy auf sich hat und ob man hierdurch nun tatsächlich

Bluetooth in Privacy-freundlicher Weise einsetzen kann. Wie durchaus zu befürchten war, kann man auch mit dem Einsatz von LE Privacy nicht davon ausgehen, als Bluetooth-Nutzer nicht mehr verfolgbar zu sein. So lassen sich Bewegungsprofile auch dann noch nachvollziehen, wenn die MAC-Adresse entweder durch zufällig gewählte Adressen sowie einer häufigen Abänderung ebendieser verschleiert ist. Dies liegt insbesondere daran, dass Werte wie der *Channel Access Code* oder der *Device Access Code* einem Angreifer Eve noch genügend Informationen bereitstellen, Rückschlüsse auf den Nutzer und sein Bewegungsprofil zu geben. Anschließend haben wir uns konkreten, in den letzten Jahren bekannt gewordenen Angriffsformen auf die Bluetooth-Übertragungstechnologie gewidmet. Aus der unter dem Begriff *Blueborne* zusammengefassten Gruppe von Verwundbarkeiten, die im Jahre 2017 veröffentlicht worden sind haben wir uns mit zwei dieser Verwundbarkeiten ein wenig genauer auseinandergesetzt. Während einer der Remote-Execution-Angriffe sich eine fehlerhafte Implementierung zunutze macht, bei der in bestimmten Konstellationen zu wenig Speicherplatz für ein entgegengenommenes BLE-Paket reserviert wird, kann Mallory die zweite der in diesem Kapitel vorgestellten Verwundbarkeiten aus der Blueborne-Gruppe nutzen, indem er die ‚Just-Works'-Authentifizierung, so wie sie unter Android umgesetzt worden ist, austrickst. So haben wird verstanden, dass letztere Verwundbarkeit die Tatsache ausnutzt, dass der ‚Just-Works'-Mechanismus ohne Authentifizierung der Gegenseite ausgelöst wird. Ein derartiges Verhalten wurde in der Spezifikation billigend in Kauf genommen und in der Spezifikation sogar benannt, was als eine bewusste Verlagerung der Verantwortung weg von den Spezifikationsentwicklern hin zu den Herstellern verstanden werden kann. Mit den Angriffen über ungültige Kurven auf ECDH haben wir dann eine Kategorie von Angriffen erörtert, die einen deutlich gewiefteren Angreifer erfordern, als dies beispielsweise für obigen Angriff auf den ‚Just-Works'-Pairing-Modus notwendig ist. Solche Angriffe sind dann insbesondere auch bei der Verwendung von *Secure Simple Pairing* (SSP) durchführbar. So konnten wir nachvollziehen, dass Mallory durch eine Manipulierung einzelner Protokollnachrichten des ECDH darauf abzielt, die Ordnung der Gruppe zur Erzeugung des Schlüssels so zu reduzieren, dass auf diese Weise Rückschlüsse auf den aktuell ausgehandelten Schlüssel möglich werden. In einer ersten Variante dieser Angriffsform aus dem Jahre 2008 muss Mallory dann den chinesischen Restsatz mit aus den Antwortnachrichten von Alice abgeleiteten Kongruenzbeziehungen auflösen, um auf den geheimen Skalar von Alice zu schließen. Verblüffend ist auch, dass es bis zum Jahre 2016 gedauert hat, bis Gegenmaßnahmen zur Verhinderung dieser Angriffsform in die Bluetooth-Spezifikationen Eingang gefunden haben. Wir haben anschließend noch einen zweiten Angriff aus der Kategorie ‚Angriff über ungültige Kurven' kennen gelernt. Bei diesem Angriff aus dem Jahre 2018 wurden von Mallory die Punktkoordinaten während der Ausführung des ECDH geändert, sodass eine Manipulation aufseiten der Opfer Alice und Bob unerkannt bleibt, indem einzig die y-Koordinaten der übertragenen Punkte von Mallory geändert wurden. Dies hatte zur Folge, dass in Abhängigkeit von der von den Opfern verwendeten Skalaren ein Angreifer auf den berechneten gemeinsamen Schlüssel

schließen kann. Deutlich länger bekannt sind hingegen Seitenkanalangriffe auf ECC und ECDH, die den verwendeten Beschleunigungsalgorithmus *Double-and-Add* zur schnellen Berechnung von Punktoperationen ausnutzen. Wir haben erörtert, wie Mallory diese zur Erlangung des geheimen Skalars nutzten kann. Hierzu muss sich das Gerät allerdings in Mallorys Besitz befinden, sodass dieser den Stromverbrauch zur Ausführungszeit messen kann.

Doch zurück zu den spezifischen Angriffen auf Bluetooth und insbesondere einem Angriff, bei dem der Angreifer eben nicht im Besitz des anzugreifenden Gerätes sein muss, sondern sich nur in dessen Nähe aufhalten muss. Eine dieser unter dem Namen ‚blutendes' Bit im Jahre 2018 bekannt gewordenen Verwundbarkeiten auf ausgewählte BLE-Chips (Bluetooth-Version 5.0) haben wir erörtert. Hierbei haben wir gesehen, dass sich Mallory zunutze macht, dass das Längenfeld eines Advertisement Beacons am Radio Core und am Main Core des Opfers unterschiedlich ausgewertet wird. Mallory kann durch ein gezieltes Senden von Advertisement-Rahmen eine entfernte Programmausführung auf dem Opfergerät mittels eines Speicherüberlaufs vorbereiten und provozieren, indem er einzelne Bits von null auf eins setzt und auf diese Weise böswillig Einfluss auf den empfängerseitig zu allokierenden Speicherplatz für einen eingehenden Rahmen nimmt.

Bei der Betrachtung von Schwachstellen der NFC-Technologie haben wir nachvollziehen können, welche Möglichkeiten schludrig verwendete Paritätsbits in Kombination mit dem Antwortverhalten eines Tags für einen findigen Angreifer bieten. Dabei haben sich je nach konkret vorliegendem NFC-Standard über die Jahre einige Spielarten dieser Angriffsform entwickelt. Allen Angriffen auf ältere Tags ist jedoch gemeinsam, dass sie entweder einzelne oder alle der folgenden Schwächen nutzen: die unvorteilhafte Verwendung von Paritätsbits, die Reproduzierbarkeit von Nonces bei einer präzisen zeitlichen Wiederholung sowie dass einzelne Schlüsselstrombits nur von der Belegung weniger innerer Zustandsbits des Krypto-Bausteins abhängig sind. Diese Designschwächen nutzen sowohl der Nested-Authentication-Angriff bei dem ein Sektorschlüssel bereits vorliegen muss, oder aber der Darkside-Angriff, der genau auf Erlangung dieses ersten Sektorschlüssel abzielt. Schließlich haben wir mit dem Hardnested-Angriff aus dem Jahre 2015 einen jüngeren Angriff diskutiert, der nicht nur bei älteren Tag-Modellen zielführend ist, sondern auch bei bereits gehärteten Nachfolgermodellen auf den verwendeten Schlüssel schließen lässt. Wir haben nachvollziehen können, wie beim Hardnested-Angriff sich der Angreifer die Summeneigenschaft zunutze macht, aus der sich ein Zusammenhang zwischen dem aktuellen Chiffrat einer verschlüsselten Nonce vom Tag und dem inneren Zustand der Chiffre für einen anschließenden Brute-Force-Angriff ergibt.

Neuere Generationen von Tags verfügen hingegen über eine Reihe an zusätzlichen Sicherheitserweiterungen, die neben den oben erläuterten Angriffen auch noch weitere Angriffsformen für Mallory erschweren sollen. So haben wir zum Ende dieses Kapitels Erweiterungen wie den Proximity-Check, die Einführung von Transaktions-MACs und

die Originalitätsprüfungen aufgezeigt. Daneben haben wir den Authentifizierungsvorgang bei aktuelleren Tag-Modellen unter Verwendung von AES beschrieben. Dieser ist einem Authentifizierungsvorgang basierend auf DES oder Triple-DES sicherlich klar überlegen und damit vorzuziehen.

Literatur

1. Armis, bleeding bit – exposes enterprises access points and unmanaged devices to undetectable chip level attack. https://armis.com/bleedingbit/. Zugegriffen: 26. Juni 2019
2. Antipa, A., Brown, D., Menezes, A., Struik, R., Vanstone, S.: Valid. Elliptic Curve Public Keys **2567**, 211–223 (2003)
3. Biham, E., Neumann, L.: Breaking the bluetooth pairing – fixed coordinate invalid curve attack. Lecture Notes in Computer Science. Selected Areas in Cryptography, Bd. 11959, S. 250–273 (2018)
4. Breig, J.: Analyse von RFID und NFC Technologie. Bachelorarbeit, Hochschule Offenburg (2019)
5. Bluetooth Specification, Version 5.0, Bd. 1 Part A. https://www.bluetooth.com/specifications/bluetooth-core-specification/. Zugegriffen: 04. März. 2020
6. Certicom Research, SEC2: Recommended elliptic curve domain parameters. https://www.secg.org/SEC2-Ver-1.0.pdf. Zugegriffen: 29. Mai 2019
7. Cryptool. https://www.cryptool.org/de/. Zugegriffen: 14. Mai 2019
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithm. MIT Press, Cambridge (1990)
9. https://www.itwissen.info/BNEP-Bluetooth-network-encapsulationprotocol.html. Zugegriffen: 26. Apr. 2019
10. Dyka, Z., Alpirez Bock, E., Kabin, I., Langendörfer, P.: Inherent resistance of efficient ECC designs against SCA attacks. In: 2016 8th IFIP International Conference on New Technologies, Mobility and Security, S. 1–5. IEEE, Larnaca (2016)
11. Fritsch, H: Design of a framework for side-channel attacks on RFID-tags. https://itookthered-pill.irgendwo.org/stuff/studium/thesis/framework-sca-rfid.pdf. Zugegriffen: 11. Dez. 2019
12. Gehrmann, C., Persson, J., Smeets, B.: Bluetooth Securiy, S. 113–117. Norwood, MA (2004)
13. Green, M.: A few thoughts on cryptographic engineering. https://blog.cryptographyengineering.com/. Zugegriffen: 29. Mai 2019
14. Kasper, T., von Maurich, I., Oswald, D., Paar, C.: Cloning cryptographic RFID cards for 25$. In: 5th Benelux Workshop on Information and System Security, WiSec (2010)
15. Lester, S., Stone, P.: Bluetooth LE – increasingly popular but still not very private. https://www.contextis.com/en/blog/bluetooth-le-increasingly-popular-still-not-very-private. Zugegriffen: 16. Sept. 2019
16. Loveless, M.: Understanding bluetooth security, Decipher security blog. https://duo.com/decipher/understanding-bluetooth-security. Zugegriffen: 16. Sept. 2019
17. Meijer, C.: Ciphertext-only cryptoanalysis on hardened mifare classic cards extended. Master's thesis, Radbound University Nijmegen (2015). https://pure.tue.nl/ws/portalfiles/portal/46945242/855438-1.pdf. Zugegriffen: 2. Sept. 2019

18. Meijer, C., Verdult, R.: Cyphertext-only cryptoanalysis on hardened Mifare Classic cards. In: 22nd ACM Sigsac Conference on Computer and Communications Security, CCS'15, Denver, Colorado, USA, S. 18–30
19. MITRE. https://www.mitre.org/. Zugegriffen: 10. Mai 2019
20. NFC-Forum. https://nfc-forum.org/. Zugegriffen: 11. Dez. 2019
21. NIST: FIPS Publication 186-2: Digital Signature Standard (DSS), Januar 2000 und Change Notice 1, Oktober 200
22. Paar, C., Pelzl, J.: Understanding Cryptography, A Textbook for Students and Practitioners. Springer, Heidelberg (2010)
23. Seri, B., Vishnepolsky, G.: The dangers of Bluetooth implementations: unveiling zero day vulnerabilities and security flaws in modern Bluetooth stacks. Armis Whitepaper, 1–41 (2017). Zugegriffen: 04. März. 2020
24. Vetter, L.: Bluetooth security in the era of IoT. Master thesis, Hochschule Offenburg (2019)

Verwundbarkeiten innerhalb der Mobiltelefonie

5

Verwundbarkeiten innerhalb der Mobiltelefonie haben eine lange Tradition und es kann sicherlich nicht der Anspruch eines allgemein gehaltenen Buches zu Mobile Security sein, diese auch nur annähernd vollständig und erschöpfend zu diskutieren. Es ist daher unser Bestreben, hier recht neue Entwicklungen aufzuzeigen, die mit der Verwendung von Smartphones für zellularen Mobilfunk Eingang in die lange Liste der Verwundbarkeiten von mobilen Systemen gefunden haben. Gleichzeitig wollen wir an dieser Stelle eine Erörterung der Sicherheitsarchitektur des 5G-Standards wagen, auch wenn diese in gewisser Weise eher spekulativ ist und vermutlich besser mit dem Begriff der Bedarfsanalyse zu umschreiben wäre. Neben den ausgewählten Schwachstellen auf dem Gebiet des zellularen Mobilfunks wollen wir auch ausgewählte Schwachstellen zur zweiten Säule der Mobiltelefonie, der schnurlosen Telefonie behandeln. Denn mit DECT ist ein recht alter, aber zu großen Teilen immer noch weltweit verwendeter Standard sehr einfach zu korrumpieren, der damit über ein durchaus hohes Schadenspotenzial verfügt.

5.1 Verwundbarkeit von DECT

5.1.1 Einleitende Bemerkungen zur Verwundbarkeit von DECT

Mc Hardy, Schuler und Tews haben schon vor einigen Jahren einen eleganten [1] Weg aufgezeigt, die verschlüsselte Kommunikation von schnurlosen Telefonen aufzubrechen. Mehr als 800 Mio. Geräte weltweit nutzen den Standard *Digital Enhanced Cordless Telecommunication*s (DECT). Damit ist DECT einer der am weitesten verbreiteten Standards für schnurlose Telefone mit kurzer Reichweite. Während europäische Systeme im Bereich von 1880 bis 1900 MHz agieren, umfasst der Bereich für Nordamerika 1920 bis 1930 MHz.

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_5

Der DECT-Standard enthält Komponenten für die Umsetzung von Verfahren zur Gewährleistung der Authentizität sowie zur Gewährleistung der Vertraulichkeit. Für ersteres hat sich der Begriff *DECT Standard Authentication Algorithm,* kurz DSAA, etabliert, letzteres erfolgt mittels der *DECT Standard Cipher* (DSC). Beide Verfahren, sowohl DSAA als auch DSC, werden nur denjenigen Herstellern vorgelegt, die ein Vertraulichkeitsabkommen (engl. *Non-Disclosure Agreement* (NDA)) unterzeichnet haben, in dem sie versichern, die dort enthaltenen Details der Verfahren nicht zu verbreiten.

Das Verfahren DSAA wird für das erstmalige Bekanntmachen eines Telefons mit der Basisstation verwendet sowie zur Authentifizierung zwischen dem schnurlosen Telefon und der Basisstation. Des Weiteren erfolgt über DSAA die Ableitung des Sitzungsschlüssels für die DECT Standard Cipher (DSC) aus dem *User Authentication Key (UAK).* Der DSC-Sitzungsschlüssels erhält den Namen *Cipher Key (CK).*

DSC ist eine Stromchiffre zur Verschlüsselung der Sprachdaten sowie einiger Kontrolldaten. Auf bekannte Weise werden ein Initialisierungsvektor (*IV*) sowie ein Sitzungsschlüssel *(CK)* zur Erzeugung eines Schlüsselstromes verwendet. Zur Verschlüsselung gelangen die Nutzdaten, im Falle von DECT sind dies die Sprachdaten, sowie Teile des Kontrollverkehrs wie die gewählte Telefonnummer.

Die ersten Angriffe auf DECT wurden bereits 2008 veröffentlicht. Mittels eines Reverse-Engineering-Ansatzes gelang es den Sicherheitsexperten, die DSAA-Authentifizierung zu brechen. Mithilfe eines passiven DECT-Sniffers unter Verwendung einer DECT-PCMCIA-Karte können Gespräche mitgeschnitten und abgehört werden. Da viele ältere DECT-Telefone und Basisstationen jedoch gar keine Verschlüsselung aktiviert haben, reicht dieses Vorgehen oftmals bereits aus. Neuere DECT-Telefone hingegen verlangen die Verschlüsselung der zu übertragenden Daten. 2009 wurde dann ein Angriff auf DSC zur Aufdeckung des Schlüssels vorgestellt. Hierzu benötigt der Angreifer 2^{15} Schlüsselströme, die unter dem gleichen Schlüssel, aber verschiedenen *IV*s gebildet wurden.

Es werden zwei Möglichkeiten erörtert, an solch eine Anzahl von Schlüsselströmen zu gelangen. Die erste Möglichkeit besteht darin, eine Situation herbeizuführen, in der von der Basisstation hin zum Telefon nur Stille übertragen wird. Dies kann erreicht werden, indem eine Sprachnachricht eingeht und der Anrufer eine lange Nachricht hinterlässt. Aufgrund der in eine Richtung übertragenen ‚Stille' ist der Klartext der Nachricht bekannt. Die Sicherheitsexperten haben nun aufgezeigt, wie der Schlüsselstrom innerhalb von schätzungsweise 10 min zu erlangen ist. Die zweite Möglichkeit nutzt den Umstand aus, dass die meisten der sich im Einsatz befindlichen DECT-Telefone die Dauer des Gesprächs auf dem Display anzeigen. Solch ein Zähler zur Anzeige der aktuellen Gesprächsdauer wird wieder und wieder pro Sekunde über den Kontrollkanal versendet. Im Falle des Gesprächsdauer-Zählers erfolgt dies über den sogenannten C-Kanal mittels einer Nachricht im C-Format. Durch dieses Verhalten auf dem C-Kanal ist die Struktur der Daten auf dem C-Kanal mehrheitlich konstant und vorhersagbar. Damit bilden diese Daten eine ideale Grundlage, um aus der verschlüsselten Nachricht den Schlüsselstrom zu extrahieren.

Der Beitrag von Mc Hardy, Schuler und Tews [1] in dieser Forschungsrichtung ist nun das Aufzeigen eines *Replay-Angriffs,* bei dem Nachrichten wiederholt eingespielt werden, um Rückschlüsse auf den Schlüsselstrom ziehen zu können. Der Angreifer benötigt hierzu keine besondere Rechenleistung. Die Forscher führen weiter an, für einen x Sekunden langen Anruf 2,8x Sekunden zu benötigen, um diesen zu brechen.

5.1.2 Das DECT-Protokoll und seine Übertragungstechnologie

Der DECT-Standard spezifiziert eine Vielfalt an Features, die zum Verständnis des hier geschilderten Angriffs als zweitrangig zu bewerten sind und an dieser Stelle daher getrost übergangen werden können. Allerdings bleibt auch festzuhalten, dass auf einem herkömmlichen schnurlosen Telefon für den gewöhnlichen Nutzer typischerweise nur eine recht kleine Untermenge dieser Eigenschaften bereitgestellt ist. Ein DECT-Netzwerk setzt sich zusammen aus einer oder mehreren Basisstationen, dem sogenannten DECT Fixed Part *(FP),* sowie einer Anzahl schnurloser Telefone, dem sogenannten DECT Portable Part *(PP).* Das DECT-Protokoll selbst besteht aus fünf Schichten: der physikalischen Schicht, der MAC-Schicht, der Data-Link-Control-Schicht, der Netzwerkschicht sowie zuoberst der eigentlichen Sprach- und Audiokodierung. Dabei erfolgt die eigentliche Verschlüsselung auf der MAC-Schicht, wobei interessant ist, dass das Aushandeln der hierfür erforderlichen Sicherheitsparameter auf der Netzwerkschicht durchgeführt wird.

Zur Nachrichtenübertragung verwendet DECT das Multiplexverfahren *Time Division Multiple Access* (TDMA). Dieses Verfahren der Nachrichtenübertragung ermöglicht es mehreren Geräten quasi simultan die gleiche Frequenz zu nutzen. Für einen Rahmen ist eine Dauer von 10 Millisekunden vorgesehen, wobei dieses Zeitintervall in weitere 24 gleich lange Zeitschlitze unterteilt ist. Pro Rahmen sind die ersten 12 Zeitschlitze für Übertragungen vom Fixed Part (*FP*) zum Portable Part (*PP*) reserviert, die restlichen 12 Zeitschlitze sind Übertragungen in die entgegengesetzte Richtung, also vom *PP* zum *FP,* vorbehalten. Innerhalb jeder dieser Zeitschlitze können 480 Bit übertragen werden, wobei ein sogenanntes DECT *full slot packet* (P32) aus 424 Bit besteht. Im Einzelnen werden $32+64+320+4+4=424$ Bit für die nachfolgend noch beschriebenen Felder S, A, B, X sowie das optionale Z-Feld verwendet. Die restlichen 50 Bit bzw. 64 Bit dienen als Schutzzeitspanne zwischen den jeweiligen P32-Einheiten. Ähnlich wie bei anderen Protokollen, die über die Luftschnittstelle der nächst höheren Protokollebene einen bestätigten verbindungslosen Dienst bereitstellen, so werden auch durch dieses Protokoll Bereiche für die Präambel, den Paketkopf, die Nutzdaten sowie die Prüfsumme abgebildet. Das S-Feld enthält die statische Präambel, die vom Empfänger zur Signalsynchronisation benötigt wird. Das A-Feld enthält den Paketkopf. Darüber hinaus wird es bedarfsweise zur Übertragung von Kontrollnachrichten über logische Kontrollkanäle verwendet. DECT sieht hierfür die logischen Kontrollkanäle C, M, N, P und Q vor. Im

B-Feld werden schließlich die Nutzdaten transportiert. Die Felder X und Z beinhalten die Prüfsumme zur Aufdeckung diverser Übertragungsprobleme.

Somit ergibt sich das folgende Bild: Ein DECT-Multiframe von 160 Millisekunden besteht aus 16 Rahmen, jeder von der Länge 10 Millisekunden. Jeder dieser Rahmen ist wiederum in 24 Zeitschlitze unterteilt, wobei die ersten 12 für die Kommunikation vom *FP* zum *PP* und die restlichen 12 für die Kommunikation vom *PP* zum *FP* vorbehalten sind. Jeder der 24 Zeitschlitze bildet wiederum ein DECT Paket der Länge 0.416 Millisekunden, wobei ein DECT-Paket in die Felder S-, A-, B, X-, Z- sowie den Schutzbereich aufgeteilt ist.

Bevor wir uns nun die Sicherheitsmechanismen von DECT ein wenig genauer anschauen, noch einige Anmerkungen zu der Verwendung von Zeitzählern (engl. *Timer*). Denn hiervon macht der DECT-Standard ausgiebig Gebrauch. Insbesondere der Zähler LCE.01 wird bei dem nachfolgend erörterten Angriff eine wichtige Rolle spielen. Dieser Zeitzähler wird immer dann gestartet, wenn davon ausgegangen wird, dass keine Verbindung mehr benötigt wird, diese also beendet und abgebaut werden kann. Der Zeitzähler ist initial auf 5 s begrenzt und initiiert nach dem Herunterzählen auf null eine Beendigung der Sitzung, sofern während dieser Zeitspanne keinerlei weitere Verbindungsaktivitäten ausgehend von *FP* oder von *PP* erfolgt sind. Dies bedeutet aber auch, dass eine aufgebaute existierende Verbindung nicht zwangsläufig gleichbedeutend mit einem aktiven Anruf ist. So kann der DECT Fixed Part die aktive Verbindung nutzen, um die Anzeige auf dem Display des schnurlosen Telefons zu aktualisieren. Des Weiteren ist es wissenswert, dass bei einer aktiven Verbindung, falls keine Audiodaten zu übertragen sind, das schnurlose Telefon P32-Pakete mit 1er-Bits oder – in hexadezimaler Notation – lauter 0xff im B-Feld versendet.

5.1.3 DECT-Authentifizierung DSAA und Verschlüsselung DSC

5.1.3.1 DECT-Standard-Authentifizierung

Mit dem *User Authentication Key* (UAK) teilen das schnurlose Telefon und die Basisstation einen symmetrischen 128Bit-Schlüssel. Dieser wird zur Authentifizierung des DECT Portable Part gegenüber der Basisstation verwendet. Eine Authentifizierung der Basisstation gegenüber dem schnurlosen Telefon sieht der Standard zwar ebenfalls vor, jedoch kommt diese in der Praxis eher seltener zum Einsatz. Für die Authentifizierung des Telefons gegenüber der Basisstation wählt die Basisstation zwei zufällige Werte *RS* und *RAND_F* und sendet diese als Authentifizierungsanfrage $AReq(RS, RAND\_F)$ an das DECT Portable Part.

Innerhalb dieser Challenge-Response-Authentifizierung verwenden beide Parteien die Algorithmen $A11$ und $A12$ aus dem DSAA, die nur den Herstellern mit unterzeichneten NDA offenbart werden. Das Protokoll endet, indem das *FP* -Gerät seinen eigens berechneten Wert $XRES1$ mit dem des am *PP* berechneten vergleicht. Bei Übereinstimmung ist die Authentifizierung des schnurlosen Telefons gegenüber der Basisstation

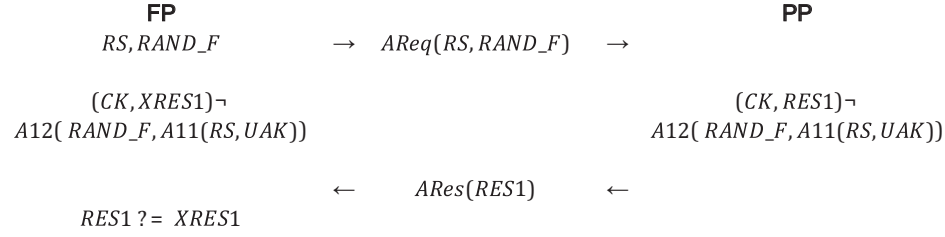

Abb. 5.1 Authentifizierung des DECT Portable Part (PP) gegenüber dem DECT Fixed Part (FP). (Quelle: eigene Darstellung in Anlehnung an [1])

erfolgreich, andernfalls nicht. Dieser Vorgang der Authentifizierung des DECT Portable Part gegenüber dem DECT Fixed Part ist in der Abb. 5.1 noch einmal veranschaulicht.

5.1.3.2 DECT Standard Cipher

Zur Einleitung der Verschlüsselung sendet die Basisstation eine Cipher-Request-Nachricht an das schnurlose Telefon. Das Telefon akzeptiert die Anfrage oder antwortet mit einer Cipher-Reject-Nachricht. Im Falle der Bestätigung durch das schnurlose Telefon sendet die Basisstation eine weitere Nachricht auf der MAC – Schicht an das Telefon. Von diesem Zeitpunkt an sind alle weiteren übertragenen Pakete verschlüsselt.

Für das weitere Verständnis ist es darüber hinaus wissenswert, dass die Basisstation kontinuierlich die Nummer des Multiframes verbreitet (engl. *Broadcast*). Jedes Paket, das von der Basisstation gesendet wird, enthält eine Rahmennummer, die sich aus der letzten verbreiteten Multiframe-Nummer sowie der Zeitspanne herleitet, die nach Eintreten dieses Ereignisses verstrichen ist.

Für die eigentliche Verschlüsselung verwendet DECT eine Stromchiffre, den DECT Standard Cipher (DSC). DSC verwendet einen 64 Bit langen Initialisierungsvektor (*IV*) sowie einen 64 Bit-Schlüssel, den *Cipher Key* (*CK*). Dabei entspricht der *IV* der Rahmennummer, welche mit 0er Bits auf 64 Bit aufgefüllt ist. Der DECT Standard Cipher generiert nun auf der Basis des Initialisierungsvektors und *CK* einen Schlüsselstrom (*CS*) der Länge 720 Bit:

$$CS \leftarrow DSC(IV, CK)$$

Die ersten 360 Bit $(cs_0, cs_1, \ldots, cs_{359})$ des *CS* dienen der Verschlüsselung von Paketen, die von der Basisstation zum Telefon übertragen werden, die restlichen 360 Bit $(cs_{360}, cs_{361}, \ldots, cs_{719})$ der Verschlüsselung derjenigen Pakete, die vom schnurlosen Telefon an die Basisstation gesendet werden. Die jeweils ersten 40 Bit, unabhängig davon, in welche Richtung gesendet wird, werden genutzt, um innerhalb des A-Feldes Nachrichten auf dem C-Kanal zu verschlüsseln. Sollte dies nicht gegeben sein, so werden die ersten 40 Bit des Schlüsselstroms einfach verworfen. Alle restlichen 320 Bit des Schlüsselstroms, die zur Übertragung von Daten in die jeweilige Richtung vorgesehen

sind, werden nun zum bitweisen XOR mit dem B-Feld genutzt. Auf diese Weise erfolgt die Verschlüsselung der Nutzdaten:

$$E(BFeld) \leftarrow BFeld \oplus (cs_{40}, \ldots, cs_{359})$$

5.1.4 Interaktive DECT-Entschlüsselung

Eine interaktive DECT-Entschlüsselung, so wie sie im Jahre 2011 von Mc Hardy, Schuler und Tews in [1] auf der ACM-Konferenz Wireless Network Security in Hamburg aufgezeigt wurde, wird nun in zwei Phasen durchgeführt. In der ersten Angriffsphase werden die Daten des Anrufs abgehört und mitgelesen. Stellt sich hierbei heraus, dass die Kommunikation unverschlüsselt erfolgte, ist der Angriff bereits erfolgreich. Dies dürfte zumindest bei neueren Geräten nicht der Fall sein, sodass die zweite Phase der interaktiven DECT-Entschlüsselung erforderlich wird.

In der zweiten Phase führt unser Angreifer Mallory eine Störsendung (Jamming) in Richtung der Basisstation durch. Dies geschieht mit dem Ziel, dass die bereits etablierte Verbindung des schnurlosen Telefons zur Basisstation abgebrochen wird. Hierzu ist es notwendig, dass Mallory mittels einer bösartigen Basisstation Rahmen auf dem gleichen Kanal und im gleichen Zeitschlitz sendet wie die eigentliche Basisstation und Mallory hierzu bedarfsweise die Frequenz sowie den gewählten Zeitschlitz ändert. Hijack als Komponente von libdect aus der Open-Source-Implementierung OsmocomDECT die die Funktionalitäten der DECT-Netzwerk-Ebene abbildet, ermöglicht die Umsetzung einer solch bösartigen Basisstation. Empfängt nun das DECT PP eine derartige Nachricht von der bösartigen Basisstation in seiner Nähe, so wechselt es u. U. zu dieser vorgeschlagenen Frequenz und diesem Zeitschlitz.

Indem eine Nachricht vom Typ LCE-PAGE-REQUEST wiederholt per Broadcast gesendet wird und das schnurlose Telefon hierauf reagiert und seinerseits eine Antwort-Nachricht sendet, kann sichergestellt werden, dass es von nun an mit der Basisstation des Angreifers verbunden ist.

Nach diesen Vorarbeiten ist es nun das Ziel des Angreifers den für den bereits getätigten Anruf verwendeten Schlüsselstrom aufzudecken. Zur Erinnerung: Dieser Schlüsselstrom wurde erzeugt unter Verwendung der nicht veröffentlichten Algorithmen $A11$ und $A12$ sowie des Schlüssels *User Authentication Key* (*UAK*), *RAND_F* und *RS*.

Der Angriff beginnt, indem Mallory dem Opfergerät erneut eine Nachricht vom Typ *AReq*() sendet, die dieses mit der Nachricht *ARes*() und der Antwort auf die Aufgabe (engl. *Challenge*) beantwortet. Nun führt unser Angreifer Mallory wiederholt folgenden Vorgang durch: Sei n die Multiframe-Nummer des ersten abgefangenen Pakets, das zu diesem Zeitpunkt noch nicht entschlüsselt ist. Dann setzt der Angreifer die Multiframe-Nummer, die von der bösartigen Basisstation verwendet wird, auf $n - (t/16)$, wobei t eine gewisse Zeitspanne ist, welche die Sicherheitsforscher Mc Hardy, Schuler und Tews wie folgt begründen: Ist t zu klein gewählt, so werden Teile zuvorderst im Schlüsselstrom

nicht aufgedeckt. Andererseits, ist t zu lang gewählt, so würde dies mit einer Performanz-Einbuße des Angriffs einhergehen. Als konkreten Wert empfehlen die Forscher, t auf die Zeitspanne zu setzen, die zwischen dem Zeitpunkt, an dem von *FP* eine Cipher-Anfrage an das schnurlose Telefon erfolgt, und dem Zeitpunkt, bis die erste Nachricht vom *PP* verschlüsselt wird, vergeht.

Die von Mallory präparierte Basisstation sendet per Broadcast nun also eine derartig geformte Multiframe-Nummer und eine Cipher-Request-Anfrage an das Opfergerät, wobei dieses auf den Erhalt dieser Nachrichten reagiert, indem es antwortet, dass es nun bereit ist, die Verschlüsselung der Pakete vorzunehmen. Das nächste vom Opfergerät gesendete Paket ist ab diesem Zeitpunkt verschlüsselt. Dabei werden der gleiche Schlüssel und der gleiche *IV* verwendet, die bereits beim ursprünglich erfolgten Anruf verwendet worden sind. Die Basisstation des Angreifers reagiert nun, indem sie von jetzt an keinerlei eigene Pakete mehr an das Opfergerät sendet. Der Angreifer wartet nun so lange, bis seitens des Opfergerätes der in Abschn. 5.1.2 erörterte Timer LC.01 heruntergezählt ist und die Verbindung aufgehoben wird. Wie bereits ausgeführt, enthalten B-Felder mit ‚geräuschlosem Verkehr' eine Folge von Hexadezimalzeichen 0xff als Klartext. Damit kann der Angreifer Mallory aber wie folgt agieren: Er verknüpft das B-Feld solch eines von der bösartigen Basisstation zum Opfergerät übertragenen Pakets und die Folge von 0xff mit XOR

$$(cs_{40}, \ldots, cs_{359}) \leftarrow BFeld_{neu} \oplus \text{0xff}$$

und erhält auf diese komfortable Art und Weise den Schlüsselstrom $(cs_{40}, \ldots, cs_{359})$. Da die exakt gleiche Bitfolge des Schlüsselstroms vom Opfergerät aber bereits für die Verschlüsselung des ursprünglichen Anrufes zur echten Basisstation verwendet worden ist, kann der Angreifer nun ein weiterer Mal XOR auf den gleichen Schlüsselstrom und das ursprünglich gesendete B-Feld anwenden und erhält somit, wie angestrebt, durch diese Interaktion mit dem Opfergerät den ursprünglich abgesetzten Anruf in Klartext:

$$\text{Klartext} \leftarrow (cs_{40}, \ldots, cs_{359}) \oplus BFeld_{alt}$$

Solch ein Angriff ist laut der Sicherheitsforscher sehr schwer durch das Opfer wahrzunehmen, da sich das Gerät weder durch Töne noch durch eine andersartige Anzeige auf dem Display ungewöhnlich verhält. Damit ist es unwahrscheinlich, einen derartigen Angriff einzig durch Beobachtung des schnurlosen Gerätes gewahr zu werden.

▶ **Wichtig**

Die vorgestellte DECT-Verwundbarkeit beruht auf der kombinierten Verwendung eines bekannten Klartextes für die Übertragung von ‚Stille' (0xff) und der Eigenschaft der XOR-Operation. Hierdurch lässt sich der Schlüsselstrom $(cs_{40}, \ldots, cs_{359}) \leftarrow BFeld_{neu} \oplus \text{0xff}$ermitteln und auf das zu entschlüsselnde Chiffrat anwenden:

Klartext $\leftarrow (cs_{40}, \ldots, cs_{359}) \oplus BFeld_{alt}$. Die Verwundbarkeit resultiert aus einer **fehlerhaften Spezifikation.**

5.1.5 War da nicht was?

Der aufmerksame Leser wird festgestellt haben, dass der gut verstandene, perfekt balancierte Krypto-Baustein XOR ein weiteres Mal, neben WEP, gebrochen wurde. Dies war möglich, indem die jeweiligen Protokollspezifikationen von WEP und DECT einem Angreifer Möglichkeiten bieten, an weiterführende Informationen zu den eingehenden Klartexten zu gelangen und diese geschickt zur Erlangung des Schlüssels bzw. früher verwendeter Klartexte zu nutzen. Übrigens: In beiden Sicherheitsteams, die das Brechen von WEP/WPA und DECT beschrieben haben, hat Erik Tews seine Expertise mit eingebracht, wobei zwischen der Veröffentlichung des Angriffs auf DECT und des Angriffs auf WEP/WPA einige Jahre ins Land gegangen sind. Das heißt, spätestens nachdem WEP gebrochen war, hätten die Entwickler von DECT vermuten können, dass auch DECT erfolgreich angreifbar sein könnte.

5.1.6 DECT-Verschlüsselung mit AES

In ihrer Arbeit zur interaktiven DECT-Entschlüsselung haben die Autoren Gegenmaßnahmen aufgezeigt, wie das wiederholte Verwenden von Zufallszahlen oder die zwingend erforderliche Authentifizierung auch der Basisstation gegenüber dem schnurlosen Telefon. Weitere Möglichkeiten wären, die Verschlüsselung zu unterbinden, sofern sie aufgrund des Sendens von 0xff nicht benötigt wird. Auch eine restriktivere Handhabung der Multiframe-Nummer wäre ratsam.

Zwei Jahre nach Veröffentlichung des geschilderten Angriffs hat das verantwortliche Standardisierungsgremium das *European Telecommunications Standards Institute* (ETSI) im Jahre 2013 dann bekanntgegeben mit DSC2 den DSC-Nachfolger zu spezifizieren. DSC2 verwendet nun die Blockchiffre AES mit der Schlüssellänge 128 Bit. Allerdings bleibt an dieser Stelle festzuhalten, dass eine vollständige Marktdurchdringung dieser Geräte noch nicht gegeben ist, sodass der geschilderte Angriff noch für eine signifikante Anzahl an genutzten Geräten erfolgreich durchzuführen ist.

5.2 Verwundbarkeit über das Baseband-Modem

Wir machen nun in gewisser Weise eine Zeitreise, weg von den schnurlosen Telefonen, wie wir sie im letzten Abschnitt behandelt haben, hin zu modernen Smartphones, die mit einem eigenen mobilen Betriebssystem ausgestattet sind und auch beim Telefonieren in den eigenen vier Wänden gerne genutzt werden. Unter dem Namen (U)SimMonitor haben im Jahre 2015 die Sicherheitsforscher Christos Xenakis und Christoforos Ntantogian [2] eine Malware vorgestellt, die auf Smartphones abzielt, die entweder mit dem mobilen Betriebssystem Android oder iOS ausgestattet sind. Hierbei nutzen sie die Verwundbarkeit des *Baseband-Modems.* Sie zeigen auf, wie Credentials und sensible Informationen,

wie IMSI, TMSI, verschiedene Schlüssel oder Positionsinformationen von der SIM/USIM Karte zu extrahieren sind. Alle verschlüsselten Telefonate können auf diese Weise entschlüsselt werden bzw. es kann ein Bewegungsprofil des Nutzers erstellt werden. Auch verschlüsselter Internetverkehr kann auf diese Weise mitgelesen werden.

Das Pikante daran: Diese Malware betrifft alle zellularen Mobilfunktechnologien, also neben GSM und UMTS insbesondere auch LTE. Dieser Mobilfunkstandard wurde seit 2013 von den Mobilfunknetzbetreibern besonders in Ballungsräumen stark ausgebaut.

(U)SimMonitor ist eine Malware, die das Baseband-Modem des mobilen Gerätes angreift. Da der Nutzer des Gerätes keinerlei Beeinträchtigung der eigentlichen Funktionalität des Gerätes wahrnimmt, erfährt er in der Regel nichts von diesem Angriff. Allerdings muss das Opfer erst einmal animiert werden, die Malware zu installieren und auszuführen. Der bei weitem häufigste Ansatz dürfte das Einschleusen der Malware-Funktionalität in Form eines Trojaners in eine legitime App für Android oder iOS sein. Frei verfügbare Binding-Werkzeuge unterstützen diesen Vorgang.

Laut [2] besteht die Malware aus Komponenten zur Erfassung von Ereignissen, zum Sammeln und Parsen von Daten sowie zum Hochladen der Daten mittels einer mit SSH gesicherten Verbindung auf einen Server. Anschließenden werden die so gesammelten Daten auf dem Gerät gelöscht. Die eigentliche Kommunikation mit dem Baseband-Modem erfolgt über ein Linux-Shell-Script, das durch SimMonitor.apk aufgerufen wird. Das Script setzt nun einzelne Befehle des AT-Befehlsatzes ab, um hierüber sensible Informationen von dem Baseband-Modem zu erlangen. Dabei ist der AT-Befehlssatz ein sich über Jahre etablierter industrieller Standard zum Konfigurieren von Modems.

Zur Ausführung der für den Angriff erforderlichen AT-Befehle benötigt (U)SimMonitor jedoch erweiterte Zugriffsrechte auf die Ressourcen des Opfergerätes, sogenannte Root-Privilegien. Doch für gewöhnlich werden diese von Android und iOS nicht so ohne Weiteres gewährleistet. Daher wird noch ein *binary* benötigt welches das Gerät nach bekannten Verwundbarkeiten durchleuchtet mit dem Ziel die Rechtevergabe des Opfergerätes auszuweiten. Dieser ausführbare Code ist oftmals mittels Verschleierungstechniken bearbeitet (obfuskiert), um statische Codeanalyse zu erschweren und gängige Antiviren-Software zu umgehen bzw. auszutricksen. Wir werden in Abschn. 7.5 die Problematik der Rechteausweitung bei mobilen Betriebssystemen noch einmal gesondert behandeln. Ebenso widmen wir uns an späterer Stelle in Abschnitt 10 den Techniken der Obfuskierung und der Deobfuskierung von Apps.

Nun kann die Malware ausgewählte AT-Befehle zur Konfiguration von Modems verwenden. Dabei werden für die Malware (U)SimMonitor vorrangig solche AT-Befehle verwendet, die zum Lesen von Daten von Interesse sind. Beispielsweise wäre das AT-Kommando zur Erlangung der *International Mobile Subscriber Identity* (IMSI) der Befehl

$$\mathrm{AT} + \mathrm{CSRM} = 176, 28423, 0, 0, 3$$

aus der Klasse der erweiterten AT-Befehle. Hierbei bedeutet ‚176' das Lesen von Daten. Die Folge 28423 besagt, dass die IMSI auszulesen ist. Mit der Folge 28448 würde beispielsweise der Schlüssel zum Verschlüsseln ausgelesen werden können. Die weiteren Werte 0,0,3 geben Informationen über Offset bzw. die Anzahl an Byte, die durch den Befehl gelesen werden sollen. Ähnliche AT-Befehle verwendet (U)SimMonitor zum Auslesen der *Temporary Mobile Subscriber Identity* (TMSI), der LAI und weiterer Schlüssel.

Eine Schwierigkeit bei der Umsetzung der Malware, so die beiden Sicherheitsforscher Xenakis und Ntantogian, bestand darin, dass der RIL-Daemon, der als Prozess im Hintergrund eines Android-Gerätes läuft, immer nur als Schnittstelle zum Baseband-Modem für genau einen Prozess dienen kann. Mit dieser Einschränkung aber ist der (U) SimMonitor-Prozess nicht in der Lage zeitgleich mit dem Android Prozess seinerseits AT-Befehle abzusetzen. Die Sicherheitsforscher haben sich für eine Umsetzung entschieden, bei der mithilfe einer Payload der RIL-Daemon immer terminiert wird, kurz bevor ein AT-Befehl abgesetzt wird, und direkt nach der Antwort des Modems auf ein AT-Kommando wieder gestartet wird.

Das Vorhandensein solch einer Kategorie von Malware für Smartphones führt uns eindrucksvoll vor Augen, wie schwer oder eher wie aussichtslos die Bestrebungen sind eine vollständig abgesicherte Architektur für ein Smartphone zu erzielen.

▶ Die geschilderte Verwundbarkeit, die von einer Malware wie (U)SimMonitor ausgeht besteht darin, dass über den AT-Befehlssatz zur Konfiguration von Modems die Malware sensible Informationen wie Schlüsselmaterial zum Entschlüsseln des Telefonats oder den Standort des Nutzers aus dem Baseband-Modem auslesen kann. Dies ist weder ein Spezifikations- noch ein Implementationsfehler. Die Verwundbarkeit ergibt sich aus einer im Vorfeld erzwungenen **Rechteausweitung** oder einer zu **laxen Rechtevergabe** auf dem Smartphone des Opfers.

5.3 5G-Sicherheitsarchitektur

Im Juli 2019 hat die Bundesnetzagentur die Frequenzblöcke der 5ten Mobilfunkgeneration (5G) für Deutschland nach einem dreimonatigen Bieterverfahren an die vier Provider Deutsche Telekom, Vodafone, 1&1 Drillisch, sowie Telefónica vergeben. Auch wenn einzelne der Bieter sich daraufhin beklagten, für die einzelnen Frequenzblöcke doch einen beachtlichen Betrag auf der Sollseite verbuchen zu müssen und diesen nunmehr nicht für den Netzausbau zu Verfügung zu haben, so kann dieses Datum sicherlich als der Startschuss für den Ausbau des 5G-Netzes in Deutschland gelten. Zur selben Zeit nehmen die nicht zuletzt durch die USA forcierten Debatten zu, inwieweit man einem chinesischen Konzern wie Huawei in den Ausbau des hiesigen Netzes einbinden darf und sich damit anfällig gegenüber Hintertüren, Spionage und Sabotage aus Richtung

China macht. Im Oktober 2019 kam dann die Entscheidung aus dem Kanzleramt, dass der Netzausrüster Huawei trotz aller Bedenken doch in den Aufbau des deutschen 5G-Netzes involviert sein darf.

Sicherlich ist die Frage, welche Netzausrüster bei der Gestaltung der 5G-Sicherheitsarchitektur als vertrauenswürdig einzustufen sind, als eine der essentiellen Fragen der organisatorischen Sicherheit in diesem Kontext zu bewerten und vermutlich von entscheidender strategischer Bedeutung für einen abhörsicheren oder auch sabotagefreien Betrieb. Es zeigt einmal mehr, dass technische Sicherheitslösungen immer nur eine Säule neben der organisatorischen Sicherheit abdecken können. Nichtsdestotrotz kann es im Rahmen dieses Buches nur unser Bestreben sein, neben diesen übergeordneten Fragestellungen, welche Ausrüster hinzuzuziehen sind, einige generelle technische Fragen hinsichtlich einer zu etablierenden 5G-Sicherheitsarchitektur zu erörtern. Dazu wollen wir zunächst die mit der 5G-Technologie anvisierten Anwendungsfelder betrachten. Denn nur wenn die Sicherheitsarchitektur die angestrebten Anwendungsfelder gegen verschiedene bekannte Angriffsformen absichert und gleichzeitig ein verlässlicher und verfügbarer Dienst bereitgestellt ist, kann diese als gelungen und passgenau bewertet werden.

5.3.1 Einleitung und Anwendungsfelder

Die vielfachen Anwendungsfelder der 5G-Netzwerkarchitektur mit maximalen Datenraten von bis zu 1250 MB/s lassen sich zu drei übergeordneten Anwendungsklassen gruppieren:

- *Enhanced Mobile Broadband* (eMBB) mit bis zu einigen Gbps in einigen Gebieten
- *Massive Machine Type Communications* (mMTC) mit der Möglichkeit der drahtlosen Kommunikation für etliche Millionen von energiebeschränkten digitalen Sensor- und Aktuator-Geräten
- *ultra reliable Machine Type Communications (uMTC)* mit einer garantierten Ende-zu-Ende-Latenzzeit unter fünf Millisekunden und einer Verfügbarkeit von 99,999 %. Gerade im Hinblick auf eine Vehicular-to-any-(V2X-)Kommunikation von Fahrzeugen mit der sie umgebenden Infrastruktur wurde die Anwendungsklasse uMTC berücksichtigt.

Durch die Untergliederung typischer Anwendungen, die vom 5G-Netzausbau profitieren sollen, in diese drei Anwendungsklassen, wird der unterschiedliche Bedarf an Datenraten, Kapazitäten, Geschwindigkeit und Sicherheit schon ein wenig offenkundiger. Als beispielhafte Vertreter für die Anwendungsklassen eMBB, mMTC und uMTC werden in dieser Reihenfolge oftmals das Video-Streaming genannt, Anwendungen, die (Temperatur)-Sensoren nutzen sowie diverse Smart-Grid-Applikationen.

Dabei sind zur Zeit des Erscheinens dieses Buches noch längst nicht alle Details und Anforderungen zum 5G-Standard verstanden oder gar umgesetzt: In Europa fördert die

EU-Kommission seit Juli 2018 drei Forschungsprojekte namhafter Industriepartner zum Thema 5G Ende-zu-Ende Verbindungen. Die Projekte *‚5G EVE' – European Validation platform for Extensive trials, ‚5G-VINNI' – 5G Verticals INNovation Infrastructure* sowie das Projekt *‚5GENESIS' – 5th Generation End-to-End Network, Experimentation, System Integration, and Showcasing* haben allesamt das Ziel, an verschiedenen Standorten in Norwegen, Frankreich, Deutschland, Italien, Spanien und Portugal die 5G-Architektur hinsichtlich ihrer tatsächlich messbaren Leistungsstärke bei gleichzeitigem Betrieb der verschiedenen Anwendungsklassen eMBB, mMTC und uMTC genauer zu evaluieren und weitere Empfehlungen zu formulieren. In Griechenland erfolgen darüber hinaus auch Tests von 5G-Hotspots auf Fahrzeugen.

5.3.2 Netzwerk-Virtualisierung und Netzwerk-Slicing

Innerhalb der 5G-Kommunikationsarchitektur wird konsequent unterschieden zwischen einer gemeinsamen physischen Infrastruktur und den virtuellen Netzen, die parallel auf der physischen Netzinfrastruktur verfügbar sind. In 5G werden solche virtuellen Netze *Slices* genannt. Dabei steht die software-basierte bedarfsgerechte Nutzung der gemeinsamen Ressource im Vordergrund. Über *Software Defined Networking* (SDN) wird eine Abstraktionsschicht über der physikalischen Schicht geschaffen mittels derer logisch getrennte virtuelle Netzwerke konfigurierbar sind. Laut ITU-T Y.301 kann das Slicing nun als logisch isolierte Netzwerk-Partitionierung mittels SDN aufgefasst werden, über die die einzelnen Slices in Bezug auf die konkreten Anforderungen für den hierin zu transportierenden Verkehr auszustatten sind. Dabei dürften die größten Herausforderungen hinsichtlich der Bandbreite insbesondere bei Slicing im Zusammenhang mit dem *Radio Access Network* (RAN) gegeben sein, also demjenigen Teil des Netzwerks, der für die drahtlose Übertragung zuständig ist. Für die einzelnen Slices können nun sowohl Minimalkapazitäten als auch Maximalkapazitäten konfiguriert werden. Das erklärte Ziel besteht nun darin, dass diese auch bei gleichzeitiger Nutzung anderer Slices nicht unter- bzw. überschritten werden dürfen. Da auf einem Endgerät typischerweise mehr als eine Anwendung zur Ausführung gelangt, kann jedes Endgerät in mehreren Slices eingebunden sein, um die einzelnen Anwendungen bedarfsgerecht zu bedienen.

5.3.3 Konzepte zur Erhöhung der Sicherheit

Die Arbeiten zur Sicherheitsarchitektur von 5G werden innerhalb der 3GPP-Arbeitsgruppe SA3 vorangetrieben. Dies geschieht in dem Bestreben, dass die Sicherheitsbetrachtungen und Risikoabschätzungen schon mit Beginn des allgemeinen Designprozesses eine hohe Wertschätzung und Berücksichtigung erfahren. Auch wenn dieses Bestreben aus dem heutigen Blickwinkel offenkundig sinnvoll erscheint, so war

doch gerade dieses Vorgehen bei früheren zellularen Mobilfunkstandards nicht immer gegeben. Die Entwickler des 5G-Standards werden sich dabei durchaus der Schwierigkeit bewusst sein, ein extrem leistungsstarkes System zu entwickeln mit hohen Bandbreiten und der Gewährleistung von bedarfsweise niedrigen Latenzzeiten, die darüber hinaus eben auch noch die notwendige Sicherheit im Hinblick auf verschiedene Schutzziele und Angreiferkategorien bieten soll. Dies dürfte gerade auch für die oben erörterten Anwendungsklassen mMTC und uMTC eine Herausforderung sein. So können für eine bedarfsgerechte Erhöhung der technischen Sicherheit einzelne Network Slices gesondert betrachtet werden, beispielsweise für Anwendungen aus dem Bereich uMTC. Für solch einen Slice kann durch Anwendung von Ende-zu-Ende-Verschlüsselung das Schutzziel Vertraulichkeit eingehalten werden. Die hierdurch anfallenden zusätzlichen Berechnungskosten können dann jedoch oftmals ein wenig konträr zu den Latenzanforderungen einzelner Verkehrsklassen ausfallen. Dies dürfte vermutlich ein Grund sein, warum der Standard eine Reihe von Verschlüsselungsverfahren zur Auswahl anbietet auf, die wir in Abschn. 5.3.5 noch genauer eingehen werden.

Die 5G-Sicherheitsarchitektur sieht die folgenden übergeordneten technischen Maßnahmen zur Erhöhung der Robustheit und Sicherheit vor:

Ad1 Ein wesentliches Augenmerk beim Entwurf einer 5G-Sicherheitsarchitektur ist es, das Netz gegenüber *‚Distributed Denial of Service‘*-(DDoS-)Angriffen robust und elastisch zu halten. Da in 5G-Netzen die schiere Anzahl der angebundenen Endgeräte für die verschiedensten Anwendungen gewaltig sein wird, erhöht sich die Eintrittswahrscheinlichkeit, aber auch das Auswirkungspotenzial von (D)DoS-Angriffen enorm. Grundsätzlich lassen sich hierbei DDoS-Angriffe unterscheiden, die direkt auf ausgewiesene Komponenten des Netzwerkes abzielen. Oder aber solche Angriffe, bei denen ein Angreifer vorab eine massive Anzahl von Endgeräten kapert, um danach hierüber in Form eines Botnets synchronisiert einen DDoS-Angriff auf eine 5G-Netzkomponente zu initiieren.

Ad2 Das Ausgleichen starker Schwankungen des Verkehrsaufkommens im Netz. Der 5G-Standard bietet die Möglichkeit einer prompten Skalierbarkeit bei planbaren Ereignissen mit massenhaften vorhersagbaren Nutzeraufkommen wie Konzerten oder Fußballspielen sowie bei nicht planbaren Ereignissen. Einer kurzfristigen massiven Zunahme der Nutzer kann zum einen durch eine Anpassung der Network Slices begegnet werden. Darüber hinaus ist es auch denkbar, kurzfristig weitere Antennen und Basisstationen bedarfsgerecht zu integrieren.

Ad3 Die Schlüsselübertragung soll im Vergleich zu Lösungen früherer Mobilfunkgenerationen, die beispielsweise auf dem Protokoll DIAMETER [3], der Weiterentwicklung von *Remote Authentication Dial in User Service (RADIUS)* [4], basierten, nun besser zu schützen sein.

Ad4 Das Einbinden von Basisstationen von hierfür nicht autorisierten Parteien soll deutlich besser verhindert werden können. Frühere Standards der Mobilkommunikation waren gegenüber IMSI-Catcher-Angriffen verwundbar, da sich hier nur das mobile Gerät gegenüber dem Netz authentifizieren musste und nicht auch das Netz gegenüber dem mobilen Gerät bevor es zu einer Übermittlung von Nutzdaten kommt. Der Catcher simuliert eine Mobilfunkzelle des Betreibers und ‚ermutigt' das mobile Gerät aufgrund hoher Signalstärken, eine Verbindung zu dem gefälschten Mobilfunknetz herzustellen. Nun kann das mobile Gerät beispielsweise dazu veranlasst werden, seine Daten unverschlüsselt an den Catcher zu senden. Bekannte Projekte und Werkzeuge hierfür sind OpenBTS, OsmonconBB oder OpenLTE sowie IMSI-Catcher PKI 1640, deren öffentliche Verwendung allesamt untersagt ist.

Ad5 Ein Integritätsschutz zu den zu übertragenden Daten soll zwingend implementiert sein; da die Verwendung einer Integritätsprüfung für einzelne Anwendungen der Applikationsklassen mMTC und uMTC von besonders leistungsschwachen IoT-Geräten jedoch zu rechenintensiv sein könnte, sieht der Standard nur eine optionale und nicht eine verbindliche Verwendung eines solchen Integritätsschutzes vor gleichwohl dieser zwingend implementiert sein soll.

Ad6 Über die Virtualisierungstechnologie des Network Slicing soll der Verkehr einzelner Teilnehmer isoliert und bedarfsgerecht hinsichtlich einzelner Schutzziele abgesichert werden können. Generell ermöglicht Network Slicing eine bedarfsgerechte Aufteilung des Netzes, je nachdem, ob beispielsweise Anforderungen an die Bandbreite oder die Latenz vorrangig zu berücksichtigen sind. Einzelne Slices sind so gegenüber anderen höherwertig zu berücksichtigen und deren Pakete beim Weiterleiten an zwischengeschalteten Netzkoppelelementen zu priorisieren. Nebenbei ergibt sich über das Network Slicing eine gewisse Verbesserung der Sicherheit, da die Daten von Nutzern anderer Unternetzwerke logisch getrennt sind und somit erst einmal nicht einsehbar sind. Einzelne Slices, die einer besonders hohen Sicherheit bedürfen können darüber hinaus mit den entsprechenden Verschlüsselungs-, Integritäts- und Authentifizierungsprotokollen versehen werden. Die verfügbaren Verfahren hierzu sind in Abschn. 5.3.5 angerissen.

5.3.4 Authentifizierung und Schlüsselübereinkunft

Die 5G-Sicherheitsarchitektur sieht eine beidseitige Authentifizierung zwischen User Entity (UE), so wird das Mobilgerät bezeichnet, und dem Netzwerk vor. Nicht nur das Endgerät authentifiziert sich gegenüber dem Netzwerk, sondern auch das Netzwerk hat sich gegenüber dem Endgerät auszuweisen. Dies ist eine Maßnahme, um vormals mögliche Angriffe wie IMSI-Catcher zu unterbinden. In 5G wird die der Bezeichner IMSI nun durch die Begriffe *Subscription Concealed Identifier* (*SUCI*) und *Subscription*

Permanent Identifier (*SUPI*) ersetzt. Der temporäre Identifier, vormals TMSI, wird nun durch die 5G-GUTI, die *Globally Unique Temporary UE Identity* ersetzt. Wobei ‚ersetzt' eigentlich nicht der treffende Begriff ist, da die *SUPI* und die *SUCI* nun auch tatsächlich technisch auf eine andere Weise hergeleitet werden als vormals IMSI und TMSI.

Zwei Arten der Authentifizierung und der Schlüsselübereinkunft gilt es im Hinblick auf die 5G-Sicherheitsarchitektur zu unterschieden: zum einen das *Extensible Authentication Protocol – Authentication and Key Agreement* (EAP-AKA), zum anderen *5G Authentication and Key Agreement* (5G-AKA). Erstere Variante kommt immer dann zur Anwendung, wenn ein Zugang aus einem 3GPP-Netzwerk erfolgen soll, andernfalls kommt die zweite Variante zum Einsatz. In beiden Fällen sind die Parteien *User Entity* (*UE*), *Security Anchor Function* (*SEAF*), *Authentication Server Function* (*AUSF*) und *Unified Data Management/Authentication Credential Repository and Processing Function* (*UDM/ARPF*) in den Protokollfluss eingebunden. Im Wesentlichen wird für beide Formen der Authentifizierung, sowohl bei EAP-AKA als auch bei 5G-AKA, ein beidseitiges Challenge-Response-Protokoll ausgelöst, dessen Einzelheiten wir uns hier jedoch ersparen wollen. Generell ist ein Challenge-Response-Protokoll ein Authentifizierungsverfahren, welches das Wissen einer Partei überprüft, um auf dessen Identität zu schließen. Dabei schließen symmetrische Challenge-Response-Protokolle, bei denen sich zwei Parteien im Vorfeld auf einen gemeinsamen Schlüssel K verständigt haben, auf die Identität der Gegenseite, indem die Partei A einen Zufallswert $RAND$ an die Partei B sendet und B bittet, diesen mittels des gemeinsamen Schlüssels K und einer beidseitig bekannten Verschlüsselungsfunktion $E()$ zu verschlüsseln. Erhält A nun $E_K(RAND)$ von B und ist die Prüfung $RAND = D_K(E_K(RAND))$ nach Anwendung der Entschlüsselungsfunktion $D()$ erfolgreich, so wird dessen Authentifizierung als erfolgreich bewertet. Man spricht von einer beidseitigen Authentifizierung, wenn dieser Vorgang in beide Richtungen erfolgt. Dabei gilt es zu beachten, dass für eine beidseitige Authentifizierung die Aufforderungen $RAND_1$ und $RAND_2$ aus unterschiedlichen Wertebereichen stammen sollten und darüber hinaus auch zwei verschiedene gemeinsame Schlüssel verwendet werden. Der eine Schlüssel ist für die Prüfung in die eine Richtung durch *UE*, der andere Schlüssel für die Prüfung in die andere Richtung durch das Netz. Für die Einzelheiten von EAP-AKA und 5G-AKA sei verwiesen auf [5]. Dabei erfolgt die Kommunikation zwischen *UE* und *SEAF* im RAN, wobei die Kommunikation der restlichen Komponenten drahtgebunden oder teilweise auch drahtlos erfolgen kann.

Mit erfolgreicher beidseitiger Authentifizierung haben sich *UE* und das Netzwerk entweder mittels EAP-AKA oder mittels 5G-AKA nun auf den von der 3GPP so bezeichneten Ankerschlüssel K_{SEAF} verständigt. Aus diesem leitet die SEAF-Komponente dann unter Einbeziehung der *SUPI* und der sogenannten *‚Anti-BiddingDown Between Architectures'*-(ABBA-)Parameter den Schlüssel K_{AMF} ab. Weitere Details zur genauen Ableitung der Schlüssel K_{SEAF} und K_{AMF} aus weiteren Parametern finden sich im 3GPP-Standard-Dokument [5] Release 15. Auf die ABBA-Parameter werden wir im Zusammenhang mit Bidding-Down-Angriffen noch einmal eingehen.

5.3.5 Verschlüsselung und Integritätsprüfung

In 5G kommen zur Verschlüsselung sowie zur Integritätsprüfung von Daten *New Radio Encryption Algorithm* (NEA) und *New Radio Integrity Algorithm* (NIA) zum Einsatz, wie sie schon bei LTE und LTE-Advanced ihre Anwendung fanden. Die verwendeten Schlüssellängen sind hierbei 128 Bit, diese sollen aber bedarfsweise auf 256-Bit erweiterbar sein.

So verwenden 128-NEA und 128-NIA die Stromchiffre SNOW 3G, die einen Schlüsselstrom auf einen Klartextstrom bitweise mit XOR verknüpft. Zur Generierung des PRNG werden ein LFSR und ein endlicher Automat verwendet. Letzterer enthält zwei Substitutionsboxen, die in ähnlicher Form innerhalb der Blockchiffre AES verwendet werden. Auf eine fundierte Sicherheitsbetrachtung der Stromchiffre SNOW 3G möchte ich hier nicht eingehen. Allerdings hat sich in den letzten Jahren wiederholt gezeigt, dass es generell enorm ambitioniert ist, tatsächlich sichere Stromchiffren zu entwerfen. So sind die Schwächen von RC4 bekannt, SEAL ist patentiert, und hardware-basierte, ursprünglich geheim gehaltene Stromchiffren wie A5/1 und A5/2 wie sie in GSM verwendet wurden, sind in der Zwischenzeit längst öffentlich und als sicherheitskritisch eingestuft. So verwundert es nicht, dass innerhalb der 3GPP-Mobilkommunikation deren Nachfolger A5/3 (KASUMI) dann eine Blockchiffre ist.

Allerdings bietet der 5G-Standard mit 128-NEA2 und 128-NIA2 auch die Möglichkeit, Verfahren zur Verschlüsselung und zur Integritätsprüfung einzusetzen, die allein auf der Blockchiffre AES aufsetzen und bei einer Blocklänge von 128 Bit mit den Schlüssellängen 128, 192 sowie 256 Bit zu verwenden sind. Für die Betriebsmodi, welche die Art der Verknüpfung der so entstehenden Chiffrat-Blöcke definieren, sind AES im CTR-Modus oder aber AES im cipher-based message authentication code, kurz CMAC, verwendbar. Letztere Variante ist interessant da man hierüber neben der blockweisen Verschlüsselung der Daten gleichzeitig auch deren Authentifizierung und Integrität gewährleisten kann. Da hierbei nur ein Schlüssel zum Einsatz kommt und auch nur ein Verfahren zur Verschlüsselung, Authentifizierung und Integritätsprüfung eingesetzt wird, kann dies insbesondere für ressourcenschwache IoT-Geräte vorteilhaft sein, bzw. für NetworkSlices, für die die konkurrierenden Ziele Sicherheit und Latenz zu harmonisieren sind. Dies dürfte zumindest für eine Reihe von Anwendungen aus der Kategorie uMTC der Fall sein, vermutlich aber oftmals auch für Anwendungen aus dem Bereich mMTC.

Daneben wird mit 128-NEA3 und 128-NIA3 Betreibern eine weitere Möglichkeit der Verwendung einer Stromchiffre gegeben. Mit der Variante ZUC unterstützt der 5G-Standard eine aus China stammende Stromchiffre oder, besser gesagt, einen PRNG, mit dem neben der Verschlüsselung auch ein Integritätsschutz der verschlüsselten Daten einhergehen soll. Die ursprüngliche Version ZUC-128 soll dann innerhalb der 5G-Sicherheitsarchitektur als ZUC-256 mit 256-Bit-Version verwendet werden [6]. Vor dem Hintergrund der Debatte zur 5G-Sicherheitsarchitektur, welche Netzausrüster aus welchen Regionen der Welt als vertrauenswürdig genug eingestuft werden, um am

Aufbau des 5G-Netzes in Deutschland mitzuwirken, dürfte es allerdings einem Schildbürgerstreich gleichen, wenn die Stromchiffre ZUC hierzulande eingesetzt werden würde. So bleibt abzuwarten, was genau die Bundesnetzagentur in ihren auf Anfang 2020 anvisierten Entwurf zu den 5G-Sicherheitsbestimmungen aufnimmt und welche Komponenten und Algorithmen der einzelnen Ausrüster das BSI auf der Grundlage der Sicherheitsbestimmungen der Bundesnetzagentur anschließend als unbedenklich prüft und zertifiziert.

5.3.6 Beispielhafte konkrete Angriffsszenarien

Was nun folgt, ist die Erörterung einiger möglicher denkbarer Angriffe auf 5G-Netze und im Idealfall Maßnahmen der 5G-Sicherheitsarchitektur, diesen zu begegnen. Dabei ist es offensichtlich, dass mit dem Rollout dieser Technologie über die Zeit sich sicherlich weitere und womöglich völlig andere Angriffsvektoren etablieren dürften, sodass wir uns an dieser Stelle mit den ‚üblichen Verdächtigen' begnügen. Dies bedeutet jedoch auch, dass wir uns vornehmlich auf nicht staatlich getriebene Angriffe beschränken und insbesondere auf solche, bei denen ein Ausrüster nicht Teil der Angriffskette ist.

5.3.6.1 DDoS-Angriffe

Ein konkreter DDoS-Angriff bestünde darin, einen EAP-AKA-DDoS zu initiieren, indem das Netz und einzelne Komponenten, vor allem diejenigen, welche die Funktionen *SEAF* und *AUSF* bereitstellen, sich der Verarbeitung einer massiven Anzahl von Authentifizierungsanfragen gegenübersehen. Natürlich sind (D)DoS-Angriffe vielfältigster Art denkbar und immer dort möglich, wo eine beschränkte Ressource adressierbar ist. Somit sind DDoS Angriffe gegen das Netz auf der *Signalisierungsebene* (Authentifizierung, Mobilitätsunterstützung, Verbindung, Bandbreite, Signaldaten jeglicher Form), auf der *Nutzerebene* (Kommunikation von Nutzdaten), auf dem RAN, aber natürlich auch auf den eingangs erwähnten logischen Ressourcen der Virtualisierungsinfrastruktur möglich.

So kann eine Überlast der Signalebene durch eine temporäre Überlast durch die Infizierung einer massiven Anzahl von M2M- oder IoT-Geräten herrühren welche zeitgleich versuchen sich mit dem Netz zu verbinden. Befinden sich diese Geräte noch dazu in derselben geografischen Region, dann kommt es fast zwangsläufig zu einer Überlast auf dem RAN aufgrund massiv verursachter Signaldaten. Zur Einschränkung eines derartigen Risikos könnten Verfahren der Anomalie-Erkennung eingesetzt werden, wobei die generelle Problematik bestünde, ein akzeptables Maß an *false-positive* und *false-negative* Raten zu erlangen. Aber auch solche Verfahren, die bei Überlast dazu dienen, einzelne Verbindungsanfragen zu filtern, um für die restlichen Verbindungsanfragen einen verfügbaren Dienst aufrechtzuerhalten, können wiederum das Ziel von DDoS-Angriffen sein. Schafft es ein Angreifer solch eine Komponente glauben zu lassen, dass ein DDoS-Angriff ansteht, so wird die Komponente zur Anomalie-Erkennung von sich

aus eine Minderung der Verfügbarkeit des Dienstes initiieren, was in seiner Auswirkung wiederum der eines (moderaten) DDoS-Angriffs gleichkommt.

Es ist also nur zielführend, Maßnahmen zur Eindämmung von DDoS, beispielsweise durch Anomalie-Erkennung massiver gleichzeitig eintreffender EAP-AKA-Aufrufe [7], so einzusetzen, dass sie den angegriffenen Dienst *graduell* herunterfahren und nicht *abrupt* den Dienst vollständig unerreichbar machen.

Man muss sich klarmachen, dass DDoS in 5G einen sehr hohen Stellenwert haben wird, schon allein aufgrund der Tatsache, dass eine massive Anzahl von M2M- und IoT-Geräten durch *Bulk-Konfiguration* oftmals nur sehr schlecht konfiguriert sein werden und deren Zugriffskontrolle auf die Ressource Gerät damit marginal sein dürfte. Damit sind solche Endgeräte sehr leicht aus der Ferne umkonfigurierbar mit dem Ziel, deren Fehlverhalten für einen massiven kontrollierten DDoS auf die Netz-infrastruktur vorzubereiten.

Auf der anderen Seite lässt sich jedoch zweifelsohne feststellen, dass die Kommunikationsmuster einer alleinigen Kommunikation zwischen digitalen Geräten ohne jegliche Nutzerinteraktion so wie dies ja in M2M- und IoT-Szenarien mehrheitlich der Fall ist, zu einem konsistenteren Verhalten führt. Die Muster einer Kommunikation sind regulärer und auch die Anzahl der verwendeten Kommunikationsprotokolle ist begrenzt. Dies alles ist wiederum für Verfahren der Anomalie-Erkennung vorteilhaft und dürfte die oben erwähnten *false-positive* und *false-negative* Raten bei der Erkennung von DDoS-Angriffen und anderen Anomalien deutlich verringern.

5.3.6.2 Bidding-Down-Angriffe

Bei einem Bidding-Down-Angriff versucht ein Angreifer in der Phase der Aushandlung der zu verwendenden kryptografischen Verfahren sich einen Vorteil zu verschaffen. Dazu versucht er die Parteien, die im Begriff sind, einen sicheren Kommunikationskanal zu etablieren, davon zu überzeugen eine schwächere Kombination bereitgestellter kryptografischer Verfahren zu verwenden, als eigentlich möglich wäre. Solche Angriffe sind prinzipiell immer dann denkbar, wenn in der Abfolge eines Protokolls die eine Seite, der Client, der Gegenstelle eine Liste von Auswahlmöglichkeiten von kryptografischen Verfahren übermittelt und diese dann hieraus eine Variante auswählt. Bei einem protokollkonformen Verhalten wird die Gegenstelle, der Server, immer diejenige Variante aus der Liste auswählen, die das höchste von beiden Kommunikationsparteien unterstützte Sicherheitsniveau erwarten lässt. Ein Beispiel hierfür wäre der Handshake bei TLS. Bidding-Down ist jedoch auch in 5G-Netzen denkbar, beispielsweise indem der Angreifer versucht, für die Authentifizierung und Schlüsselübereinkunft EAP-AKA prime zu unterdrücken und die Verwendung der mutmaßlich schwächeren EAP-AKA-Variante zu forcieren.

Zur Erschwerung derartiger Angriffe sieht der 5G-Standard die Verwendung des sogenannten AT_BIDDING-Attributes vor. Dieses Attribut wird nur in EAP-AKA-Nachrichten genutzt und nicht in EAP-AKA-prime-Nachrichten. Das AT_BIDDING-Attribut besteht im Wesentlichen aus einem 1-Bit-Flag, welches auf eins gesetzt ist,

wenn der Sender deutlich machen möchte, dass er tatsächlich EAP-AKA bevorzugt [8]. Da dieses Bit, wie die gesamte Nachricht, einen Integritätsschutz erhält, kann es von einem Angreifer, der nicht im Besitz des entsprechenden Schlüssels ist, nicht unerkannt geändert werden.

5.3.6.3 Angriffe durch Verwendung von AT-Befehlen

Es bleibt noch anzumerken, dass die 5G-Sicherheitsarchitektur, wie auch immer diese konkret ausgestaltet und in den einzelnen Ländern umgesetzt sein sollte, in keiner Weise gegen Angriffe mittels einer Malware wie (U)SimMonitor Abhilfe schafft. Wie bereits in Abschn. 5.2 erörtert, sind Smartphones, die bestückt sind mit den mobilen Betriebssystemen Android oder iOS, natürlich auch bei einer ausschließlichen Verwendung der 5G-Mobiltelefonie noch anfällig gegenüber dem Auslesen sensitiver Informationen mittels AT-Befehlen aus dem Baseband-Modem. AT-Kommandos sind die zentralen Befehle, die zur Kommunikation zwischen SIM-Karte und mobilem Gerät, SIM-Karte und Mobilfunknetz sowie dem mobilen Gerät und dem Mobilfunknetz verwendet werden. Dies bedeutet aber auch, dass prinzipiell jedes mobile Gerät über sein Baseband-Modem angreifbar ist. Malware, die AT-Kommandos nutzt, ist schon allein deshalb nicht zu unterschätzen, da auf nur sehr wenigen Geräten tatsächlich Antivirenprogramme installiert sind.

5.3.6.4 Angriffe durch Software-Updates

Zum Ende dieses Abschnitts über beispielhafte Angriffsszenarien wollen wir doch noch auf ein weiteres Dilemma hinweisen, bei dem der potenzielle Angreifer ein Innentäter, in diesem Fall also ein Ausrüster, mit beispielsweise staatlich getriebenen Interessen ist. Da die 5G-Technologie eine sehr stark softwaregetriebene Technologie ist, fallen hier auch vermehrt Software-Updates für verschiedenste Komponenten wie beispielsweise zur Netzwerk-Virtualisierung, mit *Security Anchor Function* (*SEAF*), *Authentication Server Function* (*AUSF*) oder *Unified Data Management/Authentication Credential Repository and Processing Function* (*UDM/ARPF*) an. Doch mit jedem Software-Update besteht die Möglichkeit, eine Hintertür (engl. *Backdoor*) in dem System zu platzieren, die das Mitlesen von Telefonaten, Maschinendaten oder dergleichen ermöglicht oder aber eventuell darauf abzielen könnte, die Kommunikation zu sabotieren. Hierzu sagt Ajit Pai in seiner Rolle als Chef der US-Telekommunikationsbehörde: „Es fällt sehr schwer, sich vorzustellen, dass eine Regierungsbehörde, egal wo, in der Lage wäre, jedes einzelne Update in Echtzeit zu überprüfen, um Sicherheitsrisiken aufzuspüren“. Gleichzeitig mehren sich die Stimmen von Telekommunikationsanbietern und Netzausrüstern, die sich sehr bestimmt gegen eine Offenlegung ihres Quellcodes aussprechen. Denn genau dies wird von Huawei erwartet, wenn es um die Prüfung von deren Komponenten geht. Für das deutsche 5G-Netz hieße dies, dass das BSI mit jedem solcher Software-Updates die Komponenten und Algorithmen der einzelnen Ausrüster von neuem auf deren Unbedenklichkeit zu prüfen hätte.

5.4 Zusammenfassung

Lassen Sie uns die in diesem Kapitel erörterten Verwundbarkeiten innerhalb der Mobiltelefonie noch einmal zusammentragen. Zunächst haben wir uns mit einem Angriff zum interaktiven Entschlüsseln von Anrufen unter Verwendung eines schnurlosen Telefons beschäftigt. Wir haben gesehen, dass auf ähnliche Weise, wie dies auch beim Brechen der WLAN-Verschlüsselung der ersten Generation möglich war, auch hier die XOR-basierte Verschlüsselung zu brechen ist. Konkret wird am Gerät ein Zustand erzwungen, bei dem keinerlei Audiodaten anliegen und das Gerät dann für eine gewisse Dauer ‚Stille' sendet. Solch ein bekannter Klartext kann nun eingesetzt werden, um den verwendeten Schlüsselstrom zu berechnen und diesen danach zur Entschlüsselung vorab gespeicherter eingegangener Telefonate zu benutzen.

Anschließend haben wir uns innerhalb der Mobiltelefonie einem weitreichenden Angriff im Zuge der Nutzung von zellularem Mobilfunk gewidmet. Ein solcher Angriff ist immer dann möglich, wenn das Opfer ein modernes Smartphone mit einem gängigen mobilen Betriebssystem verwendet. Die Malware (U)SimMonitor für Smartphones mit dem Betriebssystem Android oder iOS führt uns unmissverständlich vor Augen, dass, selbst wenn die drahtlose Übertragungstechnologie der eingesetzten zellularen Mobilfunktechnologie vollständig gegenüber Angriffen von Eve und Mallory gehärtet wäre, dies noch lange nicht bedeutet, dass die Kommunikation nicht abzuhören wäre. Denn solange es möglich ist, über den AT-Befehlssatz verschlüsselte Telefonate zu entschlüsseln, ist es fast schon unerheblich, wie gut die Daten auf der Übertragungsstrecke kryptografisch gesichert sind. Dabei kann die Vorbedingung für das Gelingen eines solchen Angriffs, nämlich dass im Vorfeld eine Rechteausweitung auf dem zu korrumpierenden Gerät erzwungen werden muss, nur als ein schwacher Trost angesehen werden. Nach meiner Ansicht dürfte die Verwendung von ausgezeichneten Befehlen des AT-Befehlssatzes in einer Malware zum Auslesen sensibler Nutzerinformationen oder zum Entschlüsseln von Telefonaten auch in zukünftigen 5G-Netzen mit ausgewiesener 5G-Sicherheitsarchitektur noch gegeben sein. Denn die Entwickler der 5G-Sicherheitsarchitektur werden argumentieren, dass diese Verwundbarkeit nur von einer Rechteausweitung auf dem Smartphone des Opfers herrührt, auch wenn dies eine Entschlüsselung von Verkehr im RAN zur Folge hat. Als weitere potenzielle Schwachstellen haben wir DDoS-Angriffe, Bidding-Down-Angriffe, aber auch Angriffe durch Software-Updates erläutert. Während die ersten beiden von einem externen Angreifer ausgehen, geht das mögliche Schadenspotenzial durch Software-Updates von einem Innentäter aus, beispielsweise einem Netzausrüster, der irrtümlicherweise als vertrauenswürdig eingestuft wurde und über Software-Updates einzelne Komponenten des 5G-Netzes böswillig sabotiert oder aber eine Hintertür zum Mitschneiden und Abhören von Verkehr an ausgezeichneten Netzkoppelelementen einbaut. Dies offenbart, dass neben allen technischen Umsetzungen von Sicherheitslösungen die Auseinandersetzung mit Abläufen rund um die organisatorische Sicherheit ebenfalls unabdingbar ist.

Literatur

1. McHardy, P., Schuler, A., Tews, E.: Interactive decryption of DECT phone calls. In: ACM Conference on Wireless Network Security, Hamburg, Germany, S. 71–78 (2011)
2. Xenakis, C., Ntantogian, C.: Attacking the baseband of mobile phones to breach the users' security. In: 7th International Conference on Cyber Conflict: Architectures in Cyberspace, NATO CCD COE Publications, Tallinn (2015)
3. Fajardo, V., Arkko, J., Loughney, J., Zorn, G.: Diameter base protocol. RFC 6733 (2012)
4. Rigney, C., Willens, S., Rubens, A. Simpson, W.: Remote Authentication Dial In User Service (RADIUS). RFC 2865 (2000)
5. GPP, 3gpp ts 33.501. Version 15.2.0 (f20) (2018)
6. Drucker, N., Gueron, S.: Fast constant time implementation of ZUC-256 on x86 CPUs. IEEE Annual Consumer Communications & Networking Conference (CCNC), USA (2019)
7. Harel, R., Babbage, S.: 5G security recommendations Package #1. NGMN Alliance (2016)
8. Arkko, J., Lehtovirta, V., Eronen, P.: Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA'). RFC 5448 (2009)

Klassifizierung und Risikoabschätzung

6

6.1 Was haben wir gelernt?

Die Analysen der einzelnen Verwundbarkeiten verschiedener Protokolle für eine gesicherte Drahtloskommunikation der vorangegangenen Abschnitte offenbaren, dass es so gut wie in jeder Phase der Entwicklung einer IT-System-Komponente zu Fehlern kommen kann, die zu einem späteren Zeitpunkt in einen konkreten Angriff münden können. So hat sich herausgestellt, dass einige Verwundbarkeiten aufgrund von fehlerhaften Spezifikationsdokumenten entstehen oder aber durch Spezifikationen, deren Ausformulierungen nicht präzise genug sind. Andere Schwachstellen und Verwundbarkeiten hingegen resultieren aus Implementierungsfehlern, zum Teil weil Funktionsaufrufe zum Allokieren von Speicherplatz veraltet sind und bei einer unsachgemäßen Wahl der Parameter zu Pufferüberläufen führen können. Oder aber, weil innerhalb der Implementierung durch den jeweiligen Programmierer die Ausgestaltung von Wertezuweisungen fahrlässig umgesetzt worden ist. Oft kommt dies dadurch zustande, weil eine dedizierte Angabe zu Wertezuweisungen in der zugrunde liegenden Spezifikation ausgespart blieb. Und wieder andere Verwundbarkeiten resultieren aus einer unvorteilhaften Abfolge der Bearbeitung einzelner Komponenten eines verwendeten Kommunikationsprotokolls, das für sich genommen unproblematisch ist, jedoch im Zusammenhang mit der Zielsetzung Vertraulichkeit, Integrität und Authentizität der Datenübertragung zur Aushebelung eben dieser Schutzziele führt. Die nachfolgende Tabelle verdeutlicht die in den vorigen Kapiteln zusammengetragenen Ergebnisse noch einmal. Dabei soll, soweit dies möglich ist, eine Unterteilung vorgenommen werden, in welcher Phase des Entwicklungsprozesses die Schwachstelle bzw. die Verwundbarkeit, die den konkreten Angriff ermöglicht, Eingang gefunden hat. Bei den Verwundbarkeiten, die aus einer unpräzisen Spezifikation resultieren, unterscheiden wir zwischen solchen, die das eigentliche Sicherheitsprotokoll betreffen, und denen, die eine Verwundbarkeit durch eine unpräzise Formulierung in

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_6

der Spezifikation eines anderen Protokolls hervorrufen. Für gewöhnlich ist dies das eingesetzte Kommunikationsprotokoll. Neben der Kategorie ‚Implementierung', bei denen die Verwundbarkeit sich aufgrund einer unsachgemäßen Umsetzung der Spezifikation ergeben hat, listen wir in der Kategorie ‚Andere' all diejenigen in den vorherigen Abschnitten besprochenen Schwachstellen und Angriffe auf, die wir im gewählten Klassifizierungsschema nicht eindeutig zuordnen können.

Einordnung der zusammengetragenen Verwundbarkeiten und Schwachstellen

Nr	Fehlerklasse	Protokoll	Angriff	Jahr	Bemerkung
1	Spezifikation (eigentliches Sicherheits-protokoll)	IEEE 802.11 WEP	C XOR C	Unbekannt	Zu kleiner IV, CRC32 nicht zur authentischen Datenübertragung geeignet
		DECT	$B\text{-}Feld_{neu}$ XOR 0xff	2011	Replay-Angriff unter Nutzung von Timern zur Ermittlung des Schlüssels
		Bluetooth	Pineapple	2017	‚Just Works'-Schutz geben MITM-Angriffe nicht notwendig
		Bluetooth SSP	Small Subgroup (Variante 1)	2008	Wähle Punkte Q auf Kurve mit geändertem Kurvenparameter b
		Bluetooth SSP, SC	Small Subgroup (Variante 2)	2018	ECDH-Kurvenparameter manipulierbar, da y-Koordinaten der öffentlichen Punkte nicht authentifiziert sind
		NFC	Nested Authentication	2009	Unvorteilhafte Verwendung von Paritätsbits sowie Antwortverhalten des Tags
		NFC	Darkside	2009	Unvorteilhafte Verwendung von Paritätsbits sowie Antwortverhalten des Tags
		NFC	Hardnested	2015	„Summeneigenschaft" als Beziehung zwischen innerem Zustand der Chiffre und Chiffrat

Nr	Fehlerklasse	Protokoll	Angriff	Jahr	Bemerkung
2.	Spezifikation (Kommunikations-protokoll)	IEEE 802.15.4	PiP	2011	Ausnutzung Reihenfolge Forward Error Correction zur Filterung von Nutzdaten
		WEP/WPA	ChopChop	2008	Ausnutzung von Informationen durch verschlüsselte ARP-Anfragen
		IEEE 802.11i WPA/WPA2	WLAN-ARP-Spoofing	2010	Ausnutzen, dass ARP-Nachrichten nur mit Gruppenschlüssel gesichert sind
		IEEE 802.11i WPA/WPA2	KRACK	2017	Ausnutzung des Zwei-Armeen-Problems beim Vier-Wege-Handshake zum Auffrischen der Schlüssel
3.	Implementierung	Bluetooth	Blueborne	2017	Überholte Funktionsverwendung von memcp()
		Bluetooth	Bleeding Bit	2018	Längenfeld von Advertisment Beacon an Radio-Core und Main-Core unterschiedlich ausgewertet
		Bluetooth/ ECDH	Seitenkanal über Double-and-Add	Unbekannt	Messen des Stromverbrauches bei Ausführung der Punktoperationen
4.	Andere	GSM/ UMTS/LTE	(U)SimMonitor	2015	Rechteausweitung, Auslesen über AT-Befehlssatz aus Baseband-Modem

Gliederung der beschriebenen Verwundbarkeiten hinsichtlich verschiedener Fehlerklassen

6.2 Risikoabschätzung

Nachdem wir nun eine Klassifizierung der bekannten Verwundbarkeiten durchgeführt haben wäre es vorteilhaft sich ein genaueres Bild über das Schadenspotenzial der vorgestellten Sicherheitslücken zu machen. Für gewöhnlich erfolgt dies mithilfe einer Risikoabschätzung. Entsprechend der griffigen Formel

„Risiko = Schwachstelle x Bedrohung“

besteht ein Risiko immer dann, wenn gleichzeitig eine Schwachstelle und eine Bedrohung existieren. Bekannte sowie etablierte Modelle mit dem Ziel einer quantifizierbaren Risikoabschätzung sind STRIDE, DREAD oder das sogenannte Common Vulnerability Scoring System (CVSS). Während STRIDE Risiken auflistet und hinsichtlich der Bedrohungen **S**poofing (Verschleierung), Vortäuschen einer falschen Identität, **T**ampering (Verfälschung), unerlaubtes Verändern von Daten, **R**epudiation (Ablehnung), Angreifer leugnet Durchführung einer Aktion, **I**nformation Disclosure (Aufdeckung von Informationen), Angreifer erhält Informationen ohne Berechtigung, **D**enial-of-Service (Dienstversagen), Anwendung oder System nicht mehr verfügbar, sowie **E**valuation of Privilege (Erhebung der Rechte), Angreifer ergaunert sich höhere Rechte, einordnet, zielt das DREAD-Risikomodell auf Aspekte, wie Schadenspotenzial und Verwertbarkeit ab. DREAD steht für

Damage Potenzial (Schadenspotenzial)
Reproducibility (Reproduizierbarkeit)
Exploitability (Verwertbarkeit)
Affected Users (Betroffene Nutzer)
Discoverability (Entdeckbarkeit)

und versucht in strukturierter Art und Weise Antworten auf die fünf nachfolgenden Fragen zu liefern:

Frage 1: Wie hoch ist der Schaden im Falle des Eintretens?
Frage 2: Wie leicht ist die Ausnutzbarkeit der Gefährdung reproduzierbar?
Frage 3: Was ist erforderlich um die Bedrohung auszunutzen?
Frage 4: Wie viele Nutzer werden betroffen sein?
Frage 5: Wie leicht ist es die Bedrohung aufzudecken?

Das Common Vulnerability Scoring System (CVSS) ist ein Verwundbarkeitsbewertungssystem, das 2005 vom National Infrastructure Advisory Council (NIAC) als Arbeitsgruppe des US-Ministeriums für Innere Sicherheit initiiert wurde. CVSS ist ein Industriestandard zur Bewertung des Schweregrades von Sicherheitslücken.

Für die Risikoabschätzungen der in diesem Buch zusammengetragenen Verwundbarkeiten und Sicherheitslücken auf mobile Systeme haben wir DREAD gewählt, da es zu quantifizierbaren Abschätzungen führt. Allerdings möchte ich auch klar herausstellen, dass ein mittels DREAD erstellter quantitativer Wert immer auch auf einer subjektiven Entscheidung beruht und bei verschiedenen Personen zu teilweise anderen Ergebnissen führen kann.

Eine Einordnung möglicher Werte in einem Wertebereich zwischen 0 und 10 für die einzelnen Fragestellungen wäre dabei wie folgt zu verstehen:

Frage 1: Wie hoch ist der Schaden im Falle des Eintretens?

Punkte	Beschreibung
0	Keiner
5	Einzelne Nutzerdaten kompromittiert
10	Ganzes System oder vollständige Datenmenge befallen

Frage 2: Wie leicht ist die Ausnutzbarkeit der Gefährdung reproduzierbar?

Punkte	Beschreibung
0	Sehr schwer, selbst für Administratoren
5	Angreifer autorisierter Nutzer
10	Nur ein Web-browser & URL ohne Authentifizierung

Frage 3: Was ist erforderlich um die Bedrohung auszunutzen?

Punkte	Beschreibung
0	Tiefere Programmier- & Netzwerkfähigkeiten oder spezielle Hardware
5	Malware im Internet vorhanden bzw. Exploit mittels Angreifer-Tools leicht erstellbar
10	Nur ein Webbrowser

Frage 4: Wie viele Nutzer werden betroffen sein?

Punkte	Beschreibung
0	Keine
5	Einige, aber längst nicht alle
10	Alle.

Frage 5: Wie leicht ist es die Bedrohung aufzudecken?

Punkte	Beschreibung
0	Sehr schwer bis unmöglich (Quellcode- oder Admin-Zugriff)
5	Durch Netzwerk-Monitoring erkennbar
9	Details schon mittels Suchmaschine recherchierbar
10	Information sichtbar im Webbrowser

In der nachfolgenden Tabelle sind nun einige der in diesem Buch beschriebenen Angriffe von drei unabhängigen Sicherheitsexperten hinsichtlich einer Risikoabschätzung mittels DREAD unterzogen worden. Dabei ergibt sich für das zweite D, also die Frage, wie leicht die Bedrohung aufzudecken ist, eine Besonderheit. Da in diesem Buch nur bekannte Angriffe beschrieben sind, argumentiert Sicherheitsexperte SE1 derart, dass er für diesen Punkt kategorisch 10 Punkte vergibt. Sicherheitsexperte SE2 erkennt dieses Problem auch, kommt aber zu dem Schluss, hierfür keine Punkte zu vergeben, und ‚tauft' seine Risikoabschätzung daher DREAD-D. Die Ergebnisse der Experten sind hier nun ungefiltert dargestellt. Dabei wurde nicht jeder Angriff von allen drei Experten bewertet. Die Tatsache, dass die einzelnen ermittelten DREAD-Werte schlussendlich

immer auf subjektiven Entscheidungen beruhen, lässt sich leicht an den unterschiedlichen Punktevergaben von SE1 und SE2 ablesen. Nichtsdestotrotz sind deren Einschätzungen im Regelfall ähnlich, was die Hoffnung nährt, mit DREAD eine seriös quantifizierbare Risikoabschätzung zu erhalten. Eine identische quantitative Einschätzung ist natürlich schon aufgrund der unterschiedlichen Vergaberichtlinien für den Aspekt‚Entdeckbarkeit' nicht möglich. Die wohl auffälligste Diskrepanz kann bei der Risikoabschätzung des WLAN-KRACK-Angriffs festgestellt werden, bei der SE1 den Wert 5,5 und SE2 den Wert 8,6 vergibt. Die Begründungen für die Punktevergaben beider Sicherheitsexperten für die einzelnen Fragestellungen wollen wir uns daher ein wenig genauer ansehen (Tab. 6.1):

Frage 1: WLAN-KRACK – Wie hoch ist der Schaden im Falle des Eintretens?

SE1 „Hängt vom angegriffenen Protokoll ab, Minimum ist hierbei Lesen und Replay-Angriff, Maximum ist hierbei Kontrolle des Traffics"

SE2 „Das Schadenspotential variiert abhängig vom angegriffenen Handshake, dessen Implementierung und dem verwendeten Protokoll zur Sicherstellung der Vertraulichkeit der Daten…Schadenspotential mit TKIP und GCMP größer…"
und weiter:
„Im schlimmsten Fall erhält der Angreifer Zugriff auf den Session Key eines Nutzers. Da jedoch der Master Key weiterhin geheim bleibt, ist nur der Client betroffen, dessen Handshake manipuliert wurde. Allerdings kann ein Angreifer über diesen Client Pakete mit schadhaftem Code an andere Mitglieder des Systems senden."

Frage 2: WLAN-KRACK – Wie leicht ist die Ausnutzbarkeit der Gefährdung reproduzierbar?

SE1 „Angreifer muss viele Pakete unterbinden, um ‚keychain' zu bekommen."

SE2 „Wenn ein Angreifer es schafft, einen Man-in-the-Middle zwischen Client und Access-Point zu kreieren, kann er beliebig oft die entsprechende Nachricht zum Ausnutzen von KRACK abfangen."

Frage 3: WLAN-KRACK – Was ist erforderlich um die Bedrohung auszunutzen?

SE1 „Angriff ist im Detail im Internet auffindbar und mit technischem Wissen durchführbar."

SE2 „KRACK kann mithilfe von Tools einfach ausgenutzt werden. Hierfür muss innerhalb des Tools lediglich die SSID des Netzwerks eingetragen werden. Verbindet sich ein Nutzer nun mit dem Netzwerk, wird KRACK automatisch ausgeführt und der Angreifer kann den Man-in-the-Middle-Angriff ausnutzen."

Tab. 6.1 Klassifizierung bekannter Verwundbarkeiten mittels DREAD-Analyse

Nr.	Fehlerklasse	Protokoll	Angriff	Jahr	D	R	E	A	D	Gesamt
1.	Spezifikation (eigentliches Sicherheits-protokoll)	IEEE 802.11 WEP	C XOR C	2004	10	9	10	2	10	8,2 **(SE2)**
		DECT	B-Feld$_{neu}$ XOR 0xff	2011	5 4	5 7	5 7	5 7	– 10	5 **(SE1)** 7 **(SE2)**
		Bluetooth	Pineapple	2017	8 10	7 8	5 9	8 4	– 10	7 **(SE1)** 8,2 **(SE2)**
		Bluetooth SSP	Small Subgroup (Variante 1)	2008	10	10	8	9	–	9,25 **(SE1)**
		Bluetooth SSP, SC	Small Sub-group (Variante 2)	2018	10	10	5	9	–	8,5 **(SE1)**
		NFC	Nested Aut-hentication	2009	8	8	5	5	–	6,5 **(SE3)**
		NFC	Darkside	2009	8	8	5	5	–	6,5 **(SE3)**
		NFC	Hardnested	2015	8	8	5	10	–	7,75 **(SE3)**
2.	Spezifikation (Kommunikations-protokoll)	IEEE 802.15.4	PiP	2011	8	3	5	6	10	6,4 **(SE2)**
		WEP/WPA	ChopChop	2008	5	10	5	4	10	6,8 **(SE2)**
		IEEE 802.11i WPA/ WPA2	WLAN ARP-Spoofing	2010	8	10	6	10	10	8,8 **(SE2)**
		IEEE 802.11i WPA/ WPA2	KRACK	2017	7 9	5 10	5 10	5 4	– 10	5,5 **(SE1)** 8,6 **(SE2)**
3.	Implementierung	Bluetooth	Blueborne	2017	10	9	10	8	–	9,25 **(SE1)**
		Bluetooth	Bleeding Bit	2018	5	3	5	3	–	4 **(SE3)**
		Bluetooth resp ECDH	Seitenkanal auf Double-and-Add	–	5	3	0	5	5	3,6 **(SE3)**
4.	Andere	GSM/ UMTS/LTE	(U)Sim-Monitor	2015	10	5	3	10	–	7 **(SE3)**

Frage 4: WLAN-KRACK – Wie viele Nutzer werden betroffen sein?

SE1 „Inzwischen wurden viele Software-Patches geliefert, Legacy-Hardware ist nach wie vor betroffen.“

SE2 „WPA2 ist der aktuelle Standard zur Sicherung von WLAN-Systemen. Solche Systeme werden sowohl im Unternehmen als auch in privaten Haushalten und öffentlichen Hotspots von einer Vielzahl von Nutzern benutzt und sind weit verbreitet. Die Lücke wurde kurz nach der Entdeckung durch Patches geschlossen, womit lediglich alte Geräte oder Geräte ohne Sicherheitsupdate betroffen sind.“

Frage 5: WLAN-KRACK – Wie leicht ist es die Bedrohung aufzudecken?

SE1 -

SE2 „Die Schwachstelle ist bekannt und öffentlich einsehbar.“

Teil III
Softwarekomponenten mobiler digitaler Geräte

Bisher haben wir uns im Rahmen dieses Buches ausschließlich mit den drahtlosen Übertragungstechnologien heutiger mobiler digitaler Geräte auseinandergesetzt. In diesem Teil des Buches wollen wir uns nun mit den wesentlichen Software-Komponenten beschäftigen, die ein modernes Smartphone ausmachen. Zu nennen ist hierbei natürlich zunächst einmal das Betriebssystem, wobei wir uns mehrheitlich auf das quelloffene Betriebssystem Android konzentrieren werden. Es ist unser Anliegen, die Sicherheitsarchitektur von Android zu verstehen, um eine bessere Einschätzung zu erlangen, was es leisten kann und was nicht. Daneben sollen uns aber auch der Aufbau und die Vorgaben zur Umsetzung insbesondere sicherheitskritischer mobiler Anwendungen, kurz Apps, interessieren. Gerade am Beispiel von mobilen Banking-Apps lässt sich sehr anschaulich verdeutlichen, welcher Aufwand getrieben werden muss, um eine Anwendung tatsächlich seriös zu härten, und was dennoch so alles schiefgehen kann, wenn man sich findigen Angreifern gegenübersieht. So veröffentlicht das OWASP Mobile Security Project [1] regelmäßig eine Liste der vorrangigen Sicherheitsrisiken für mobile digitale Geräte. Die aktuellen Top 3 auf dieser Liste sind die ungeeignete Verwendung des Gerätes, eine ungesicherte Datenablage sowie eine unsichere Kommunikation, beispielsweise aufgrund fehlender Zertifikatsprüfung bei einem TLS-Handshake zum Aufbau einer gesicherten Ende-zu-Ende-Verbindung oder aber durch das Übertragen sensibler Informationen über eine vollständig ungesicherte Verbindung. Aber auch die weiteren Punkte auf der Liste der OWASP sind nicht nur bei der Erstellung von Banking-Apps hoch relevant. Neben der Verwendung unsicherer kryptografischer Bausteine oder einer unsicheren Autorisierung verweist die OWASP auch auf die oftmals schlechte Qualität des Client-Codes, der die Gefahr von Pufferüberläufen in sich trägt. Auch die Hinweise, dass Angreifer mehr und mehr über Reverse-Engineering-Techniken verfügen, um eine App zu analysieren, oder versuchen, den Debug-Modus zu aktivieren, um über die so erhaltenen Ausgaben beim Starten der App wertvolle Informationen für einen gezielteren Angriff zu sammeln, ist gerade auch bei Banking-Apps von hoher Brisanz und wird uns

im Rahmen dieses Teils des Buches noch beschäftigen. Wir beenden Teil 4, indem wir noch einmal auf das Betriebssystem eines modernen mobilen Gerätes zu sprechen kommen und hier insbesondere auf Fähigkeiten, die wohl für die meisten seiner Nutzer weitgehend verborgen sind. Uns interessieren insbesondere solche Aktivitäten, die moderne mobile Betriebssysteme wie Android, aber auch iOS permanent permanent hinter unserem Rücken durchführen.

Literatur

1. OWASP: Mobile Top 10 2016-Top 10. https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10. Zugegriffen: 22. Okt. 2019

Das Betriebssystem Android

7

In der Tat wäre es wünschenswert, verschiedene mobile Betriebssysteme hinsichtlich ihrer Sicherheitskomponenten vorzustellen und zu vergleichen. Dies sprengt allerdings ab einem gewissen Detaillierungsgrad auch den Umfang eines Buches. So dient uns in diesem Buchteil das mobile Betriebssystem Android als Referenzsystem, wohlwissend, dass es mit iOS und, nicht zu vergessen, dem im Spätsommer 2019 erstmalig verfügbaren Betriebssystem Harmony OS von Huawei weitere mobile Betriebssysteme gibt, die durchaus eine ausführlichere Betrachtung wert wären. Ob das quelloffene Harmony OS zukünftig tatsächlich in der Lage ist, die Vormachtstellungen der Systeme Android und iOS zu gefährden, ist zum Zeitpunkt des Erscheinens dieses Buches völlig offen, jedoch keineswegs undenkbar.

7.1 Die Android-Systemarchitektur

7.1.1 Grundlagen zur Systemarchitektur

Das Open-Source-Betriebssystem Android OS hat sich über die Jahre von der Version 1.0 (Base) zur Version 10 (Stand September 2019) von einem reinen Betriebssystem für Smartphones und Tablets auch für den Einsatz in TVs, digitalen Kameras oder Car-Entertainment-Systemen weiterentwickelt. Dabei ist die Architektur des Betriebssystems in vier Schichten darstellbar: Android basiert im Kern auf dem Betriebssystem Linux, allerdings mit einer Reihe Erweiterungen wie einem Binder für die *Inter Process Communication* (IPC) sowie Techniken zur Verwaltung des gemeinsamen Speichers. Dabei ist der Binder vergleichbar mit der Umsetzung des Kommunikationsmechanismus *Remote Procedure Call* (RPC) [1] in verteilten Systemen, in Android nun aber ist er zuständig für die IPC innerhalb eines Systems, für die Kommunikation zwischen

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_7

Prozessen auf einem Betriebssystem. Damit erhält in Android ebenfalls jeder Prozess, oder genauer ein sich in der Ausführung befindendes Programm, eine Prozess-ID, die sogenannte PID. Ein aktuell laufender (Eltern-)Prozess kann einen (Kind-)Prozess erzeugen, wobei die Systemaufrufe `fork()` und `execve()` hierbei bedeutsam sind. Ersterer erzeugt einen neuen Kind-Prozess mit neuer PID als identische Kopie des ausführenden (Eltern)-Prozesses. Letzterer sorgt dafür, dass der ausführende Prozess durch das neue Programm ‚überlagert' wird. Der Prozesskontext und die PID bleiben dann die gleichen. Die mittlere Schicht der Architektur des Android-Betriebssystems beinhaltet eine Anzahl nativer Bibliotheken, beispielsweise OpenGL, libc, SQLite oder auch SSL. Neben diesen Bibliotheken enthält diese Schicht darüber hinaus die Android-Laufzeitumgebung, in der die in den Programmiersprachen C bzw. C++ geschriebenen Systembibliotheken enthalten sind, sowie die *Dalvik Virtual Machine* (DVM), die ab Version 5.0 (Lollipop) in 2014 durch die *Android Runtime* (ART) ersetzt worden ist. Beide sind an der ARM-Architektur angelehnte registerbasierte VMs, die insbesondere für den Bedarf ressourcenschwacher Geräte ausgestaltet wurden. Das darüber liegende Applikationsframework kann als eine Übersetzungsschicht verstanden werden und stellt Schnittstellen bereit über die ein App-Entwickler auf diverse Eigenschaften des Gerätes zugreifen kann. Auf dem Applikationsframework setzen dann die einzelnen Anwendungen auf. Eine Übersicht des Aufbaus der Architektur des Android-Betriebssystems stellt sich damit wie in der Abb. 7.1 aufgeführt dar.

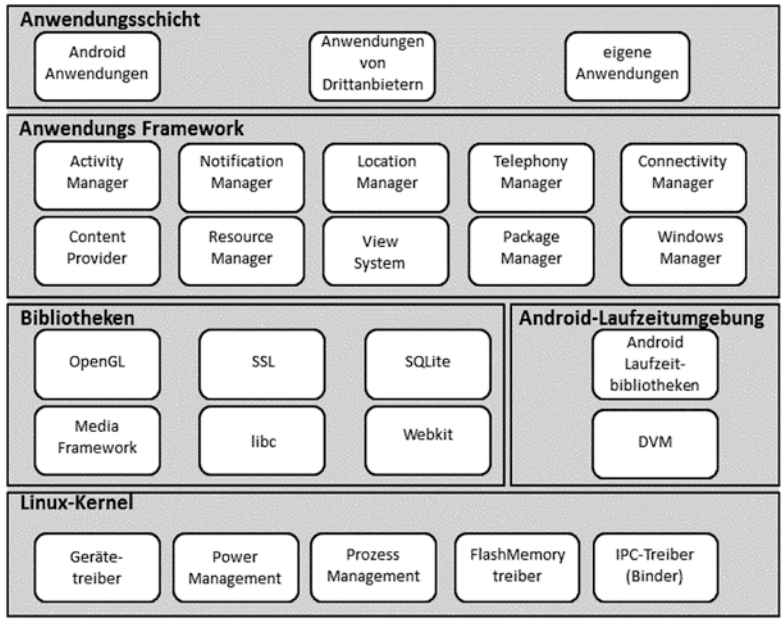

Abb. 7.1 Übersicht der Architektur des Android-Betriebssystems. (Quelle: eigene Darstellung in Anlehnung an [2])

7.1.2 Der Bootvorgang, ART und Zygote

Das Hochfahren (engl. *boot*) eines mobilen Gerätes, auf dem das Betriebssystem Android installiert ist, stellt sich nun folgendermaßen dar:

1. Der Bootloader lädt den Linux-Kernel und startet den Init-Prozess mit der PID=1.
2. Der Init-Prozess startet Linux-Daemons, wie die Android Debug Bridge (adb).
3. Init startet den Zygote-Prozess.
4. Init startet den Runtime-Prozess.
5. Der Runtime-Prozess startet insbesondere den Service Manager.
6. Runtime sendet Anfrage an Zygote, um System-Server zu starten.
7. Zygote erzeugt mit fork() einen Kindprodess und startet den System-Server als ersten Prozess in der ART (vormals DVM).
8. System-Server startet Audio und Surface Flinger (Kontrolle für Display & Audio).
9. Diese registrieren sich beim Service Manager.
10. System-Server startet Core und Hardwaredienste.
11. Diese registrieren sich beim Service Manager.

Damit lässt sich diese soeben geschilderte Boot-Reihenfolge folgendermaßen illustrieren (Abb. 7.2):

Aus dieser Abfolge von Prozessen, die allesamt aus dem Init-Prozess hervorgegangen sind, sind insbesondere der Zygote-Prozess sowie die Funktionsweise der Android Runtime (ART) beachtenswert. Im Einzelnen stellen sie sich wie folgt dar:

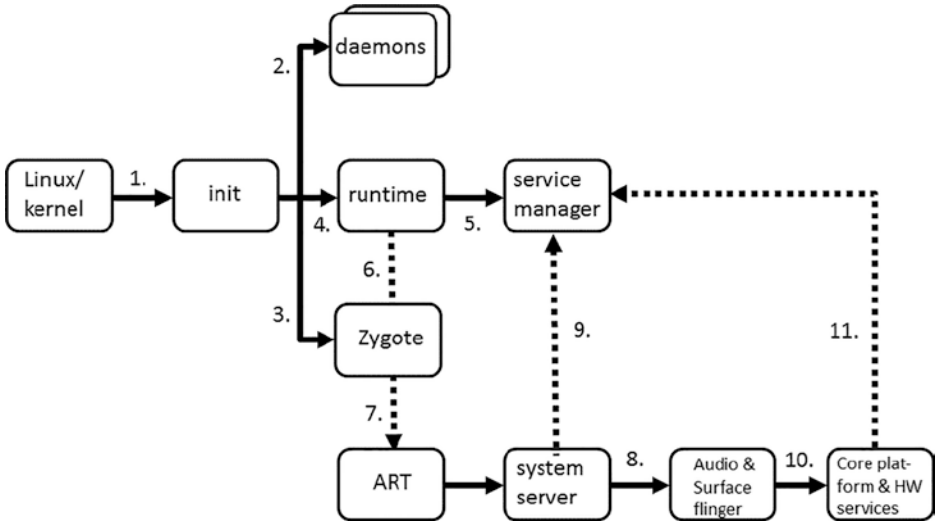

Abb. 7.2 Boot-Reihenfolge von Android beim Hochfahren eines mobilen Gerätes. (Quelle: eigene Darstellung)

Der Zygote-Prozess: Den Zygote-Prozess kann man sich als eine leere Instanz einer ART vorstellen, die erst einmal gar nichts ausführt. Dieser Prozess, dessen Name an den Vorgang der Befruchtung von Lebewesen aus der Biologie angelehnt ist, wird nun verwendet, um sich mittels des Systemaufrufes fork() zu klonen und so, wenn Bedarf besteht, eine weitere ART zu instanziieren. Ist dies geschehen, so übergibt der Zygote-Prozess die Anwendungsdaten an den neuen Prozess. Dabei teilt sich der neu ‚geforkte' Prozess den dynamischen Speicher (Heap) solange mit dem Zygote-Prozess bis eine hierin gestartete App ihre Daten erstmalig auf den Heap schreibt. Erst ab diesem Zeitpunkt erhält die ART mitsamt der App einen eigenen zusammenhängenden Speicherabschnitt für ihren dynamischen Speicher.

Android Run Time (ART): ART und vormals die DVM bilden den Hauptteil der Laufzeitumgebung und wurden eigens ausgestaltet für mobile Geräte mit vergleichsweise wenig Rechenleistung und wenig Speicher. Ab Android 5.0 wurde die Dalvik Virtual Machine durch die ART ersetzt, wobei beide kompatibel sind. Dies bedeutet, dass Anwendungen, die für Dalvik entwickelt wurden, auch in der ART laufen. In ART wurde der *Just-in-Time* -(JIT-)Ansatz durch den sogenannten *Ahead-of-Time*- (AOT-)Ansatz ersetzt. So werden im .dex-Format kompiliert, noch bevor diese benötigt werden. Diese Vorgehensweise führt zu deutlichen Geschwindigkeitsverbesserungen und Einsparungen beim Stromverbrauch. Der *Garbage Collector* als diejenige Komponente, die für eine Bereinigung der aktuell nicht mehr verwendeten Daten aus dem Speicher zuständig ist, ist ebenfalls schneller geworden. Darüber hinaus zielt der Garbage Collector bei der Freigabe von Speicherplatz auch darauf, ab einen weniger fragmentierten Speicher zu übergeben.

Speicheroptimierung: Das .dex-Dateiformat für Dalvik Executables benötigt deutlich weniger Speicher als das Standardformat. So befinden sich in einem .dex-File mehrere Klassen und nicht pro Klasse ein .class-File. Eine derartig kompakte Darstellung ermöglicht es, redundante Daten einzusparen. Als weitere Konzepte zur Speicheroptimierung sind die Umsetzung einer *Copy-on-write*-Policy sowie die Verwendung von Markierungsbits (engl. *mark-bits*) für den Garbage Collector zu nennen. Unter Verwendung der Copy-on-write-Policy schreibt eine App auf geteilte Daten, diese werden in den lokalen Heap kopiert und dort modifiziert. Mit einem optimierten Garbage Collector für jede sich in der Ausführung befindende App und der Verwendung von Markierungsbits kann der Garbage Collector feststellen, welche der geteilten Daten vermutlich noch von einer App verwendet werden und welche automatisch zu löschen sind, weil diese nicht mehr benötigt werden.

CPU-Optimierung wird durch eine Reihe von Maßnahmen erreicht: Das Auslagern (engl. *Swapping*) erfolgt byteweise. Dies führt zu einer Verbesserung der Anordnung des Speichers. Generell wird bei der Speicherverwaltung durch Auslagern im Gegensatz zum virtuellen Speicher jeder Prozess komplett in den Hauptspeicher geladen, läuft für eine gewisse Zeit, und wird danach wieder vollständig ausgelagert. Des Weiteren werden statische Links verwendet und um teure Funktionsaufrufe zu minimieren, werden native Funktionen in den eigentlichen Code intergiert (engl. *inlining*).

7.1.3 Abgesichertes Hochfahren

Die in der Abb. 7.2 dargestellte Boot-Reihenfolge wird seit der Android-Version 4.4 aus dem Jahre 2013 durch einen verifizierten Bootvorgang abgesichert. Ziel dabei ist es eine Vertrauenskette beim Start des Gerätes aufzubauen, mit der die Integrität jeglicher genutzter Ressourcen überprüft werden kann. Eine derartige Vertrauenskette beginnt bei einem *hardwaregeschützten* Vertrauensanker. Mit dem eigentlichen Bootvorgang des Gerätes einhergehend wird diese Vertrauenskette unter Einbeziehung des Bootloaders etabliert. Jede einzelne Stufe innerhalb des Bootvorgangs überprüft die Authentizität und Integrität der ihr nachfolgenden Stufe. Nur wenn die Integritätsprüfung aller dieser Stufen erfolgreich ist, wird der Bootvorgang vollständig beendet, andernfalls wird dieser abgebrochen. Zu unterscheiden sind bei der Prüfung zwei Arten von Fehlern: Software- und Hardwarefehler, die sich ohne eine bösartige Manipulation eingeschlichen haben, sowie solche die tatsächlich aufgrund von bösartigen Manipulationen einzelner Komponenten im System vorhanden sind. Mit der Version 7 verfügt das abgesicherte Hochfahren über Verfahren der *Vorwärtsfehlerkorrektor*, mit denen aufgrund einer geschickten Kodierung der Daten zumindest eine Reihe von Fehlern beim Hochfahren korrigiert werden können. Es wird also Redundanz in Form von Paritätsbits eingebracht. Hierbei lautet das übergeordnete Ziel, auch in Gegenwart derartiger Fehler den Bootvorgang erfolgreich durchzuführen [3]. Im Einzelnen werden zur *Vorwärtsfehlerkorrektor* RS-Kodes der beiden Kodierungstheoretiker Irving Reed und Gustave Solomon [4] verwendet sowie signierte Hashbäume so wie sie von dem amerikanischen Kryptografen Ralph Merkle schon Ende der 70er-Jahre in einem Patent [5] vorgeschlagen worden sind. Hinter jedem dieser Ansätze verbergen sich höchst spannende Konzepte, deren Beschreibung an dieser Stelle vermutlich doch deutlich zu weit führen würde. Anders dagegen wird verfahren, wenn die Integritätsprüfung fehlschlägt: Hier ist von einer bösartigen Manipulation einzelner Ressourcen auszugehen, sodass der Bootvorgang abgebrochen wird. Ab der Version 7 enthält Android den sogenannten *Android Verified Boot* [6, 18] der darüber hinaus Updates und Patches dahingehend einbezieht, dass ein Rollback-Schutz vorhanden ist. Das erklärte Ziel des Rollback-Schutzes ist es nun, einen möglichen Exploit zumindest nicht dauerhaft auszuführen.

7.2 Aufbau einer Android-App

7.2.1 Aufbau und Komponenten

Zum besseren Verständnis des Rechtemodells von Android (siehe Abschn. 7.4), möglicher Kommunikationswege zwischen Android-Applikationen (siehe Abschn. 7.2.2) aber auch der Problematik einer Rechteausweitung (siehe Abschn. 7.5) bedarf es solider Kenntnisse, wie eine Android-Anwendung im Einzelnen aufgebaut ist. Die einzelnen Komponenten einer App wollen wir daher an dieser Stelle in kompakter Form vorstellen:

Die Manifest-Datei *AndroidManifest.xml* muss innerhalb des APK-Archivs zwingend erforderlich vorhanden sein. Sie enthält Informationen über die App, wie beispielsweise ihren Namen und ihre ID, aber auch auf welche Art die nachfolgend vorgestellten Komponenten *Activity, Services, Content Provider* und *Broadcast Receiver* betrieben werden. In der Manifest-Datei sind auch die Rechte eingetragen, mit denen die App betrieben werden muss um tatsächlich ausführbar zu sein. Gleichzeitig enthält sie Informationen zu den erforderlichen Hardware- und Softwarespezifikationen, wie beispielsweise die Mindestversion des Android-Betriebssystems, die auf dem Smartphone installiert sein muss.

Eine *Activity* ist die zentrale Komponente, über welche die direkte Kommunikation mit dem Nutzer erfolgt. Dies geschieht in den meisten Fällen über eine Benutzerschnittstelle (engl. *User-Interface*), auch wenn dies jedoch nicht zwingend erforderlich ist. Die Activity einer App bildet häufig den Startpunkt einer Interaktion mit dem Nutzer. Dabei sind die Vorgaben zum Design der App und deren Funktionalität strikt voneinander getrennt. Die Funktionalität ist in der Methode onCreate() zu initialisieren, das Design kann deklarativ beispielsweise unter Verwendung des XML-Formats abgelegt werden.

Die Komponente *Service* ist für das Ausführen von Hintergrundaufgaben zuständig, wie beispielsweise das Abspielen von Musik oder das Laden einer Datei. Damit benötigt sie typischerweise kein User-Interface. Ihre Lebenszeit ist üblicherweise länger als die einer Activity. Hierbei ist es wissenswert, dass die Methoden aus Services anderen Komponenten zu Verfügung gestellt werden können, sowohl den Komponenten der eigenen App als auch anderer Apps. Dies kann mittels des *RPC-Binders* oder aber der *Android Interface Definition Language* (*AIDL*) erfolgen, wobei ab der Android-Version 8.0 derartige Zugriffe im Hintergrund jedoch deutlich eingeschränkt wurden, wenn die zugegriffene Applikation selbst nicht auch im Vordergrund läuft. Hierauf werden wir innerhalb des Abschn. 7.2.2 noch genauer eingehen.

Zwei andere Komponenten stehen der weiteren Strukturierung einer App zur Verfügung: der sogenannte *ContentProvider* und der *BroadcastReceiver*. Die Komponente ContentProvider dient als Speicher zur dauerhaften Ablage von Daten und ist systemweit ansprechbar über einen *Uniform Resource Identifier* (URI). Eine derartige Art der Adressierung ist dabei auch zur Kommunikation über Apps hinweg nutzbar. Die Operationen sind die üblichen Verdächtigen und umfassen die Aufrufe *insert, query, update* und *delete*. Als letzte der zu nennenden Komponenten fehlt noch der BroadcastReceiver. Diese Komponente dient dem Empfangen von IPC-Nachrichten, den sogenannten *Intents*. Dabei können gesendete Intents nicht nur von einer Partei empfangen werden, sondern von mehreren Parteien, den sogenannten *BroadcastReceivern.* Für welche anderen Apps konkrete Intents einer anderen App tatsächlich vorgesehen sind, wird in der *Manifest*-Datei dieser App über einen Filter festgelegt und dort eingetragen. In der Manifest-Datei *AndroidManifest.xml* werden darüber hinaus die Struktur der App in einem XML-basierten Format beschrieben sowie die Rechte, welche die App zur Durchführung ihrer Tätigkeiten auf dem Smartphone benötigt.

7.2.2 Mögliche Kommunikationswege zwischen Android-Applikationen

Befasst man sich ein wenig eingehender mit den möglichen Kommunikationswegen bzw. Datenflüssen zwischen Apps unter Android, so kann man schnell zu der Auffassung gelangen, dass man es hierbei mit einem *verteilten System* [7] auf genau einer Plattform zu tun hat. Diesen Gedanken möchte ich ein wenig konkreter ausführen: Apps und Systemdienste verhalten sich wie in einem verteilten System. Dabei ist es gar nicht so einfach zu definieren, was genau ein verteiltes System ausmacht. Das Aggregat aus einer ganzen Reihe von Definitionsversuchen trifft neben dem der ursprünglich intendierten transparenten Kommunikation verschiedenster Rechnerarchitekturen auch auf die Kommunikationsarchitektur innerhalb des Android-Betriebssystems zu: Das verteilte System besteht aus Komponenten, diese agieren autonom und dem Benutzer kommt es dabei vor, als hätte er es mit einem einzigen System zu tun. Dabei ist die Gewährleistung der Zusammenarbeit die Kernaufgabe von verteilten Systemen, die technisch mithilfe einer *Middleware* umgesetzt wird. Insbesondere das Semantikkonzept von *Remote Procedure Call* (RPC) [1], dass bereits im Jahre 1984 von den Forschern Andrew Birell und Bruce Nelson für Prozeduraufrufe über Rechner hinweg am Xerox-Forschungszentrum in Palo Alto entwickelt wurde, findet sich nun auch als eine Form der Kommunikation zwischen Apps auf einem mobilen Gerät wieder.

Unter Android läuft nun jede aufgerufene App in einer eigenen Instanz der Android Runtime, wobei die ART in Bezug auf die Ausführungsrechte angeht, als Prozess mit beschränkten Rechten auf dem Android-Betriebssystem zu verstehen ist. Durch das Sandbox-Rechte-Konzept hat eine sich in der Ausführung befindende Android-Applikation nur Zugriff auf ihre eigenen Daten. Soweit, so gut. Welche Kommunikationsmöglichkeiten aber hat nun eine App? Tatsächlich unterscheiden wir vier Arten der Kommunikation. Diese gestalten sich aus den einzelnen Komponenten einer App-Struktur wie folgt: Die Komponente Activity kommuniziert mit Hilfe von *Intents* und Results. Eine Komponente Service hat die Möglichkeit, über entfernte Methodenaufrufe, die oben genannten RPCs, unter Verwendung von AIDL Daten auszutauschen, während der ContentProvider mittels *Queries* und *Cursor* und der BroadcastReceiver mittels BroadcastIntents kommunizieren können.

Der offizielle Weg einer Kommunikation zwischen Apps erfolgt über Intents (Activity) oder RPC-Aufrufe (Service) mithilfe des Binders für die Interprozesskommunikation (IPC). Dieser ist in einem herkömmlichen Linux-Kernel seit 2015 vorhanden. Hierüber können Objekte oder Methoden eines Prozesses einem anderen Prozess zur Verfügung gestellt werden. Konkret handelt es sich bei den Prozessen um die Instanz einer Sandbox einer Android Runtime. Dabei ist ein wesentlicher Aspekt des RPC-Konzeptes von Birell und Nelson, dass der Aufruf entfernt ausgeführter Methoden für den aufrufenden Prozess nicht vom Aufruf lokal ausgeführter Methoden zu unterscheiden ist. Um die angestrebte Transparenz zu gewährleisten, soll der aufrufende

Prozess diese Unterscheidung auch gar nicht treffen können. Unter Verwendung der Android Interface Definition Language (AIDL) bietet der Binder nun eine Schnittstelle all derjenigen Objekte/Methoden an, die mit RPC aufrufbar sein können. Android SDK bzw. Android Studio erstellt dann auf der Grundlage dieser AIDL-Schnittstellenbeschreibung die hierfür passenden Proxy- und Stub-Klassen. Diese senden den konkreten Methodennamen und die übergebenen Parameter in einer Nachricht damit sie empfängerseitig ausgeführt und die Ergebnisse des Methodenaufrufs anschließend wieder an den aufrufenden Prozess übertragen werden.

Die Einteilung der möglichen Kommunikationswege entsprechend den einzelnen Android-Komponenten ist noch einmal in der Abb. 7.3 dargestellt.

Intents und Broadcast Intents sind abstrahierte Nachrichten und dienen dem Starten neuer Activities oder aber der Kommunikation zwischen Activities. Dabei ist, wie oben bereits erwähnt, der Aufruf entfernter Objekte von lokalen Objekten nicht unterscheidbar. Neben den expliziten Intents bestimmt bei den impliziten Intents das System, an wen diese gesendet werden. BroadcastIntents sind von mehreren Apps zu empfangen und sind hilfreich für verschiedene Ereignisse wie das Installieren weiterer Apps oder die Information über den aktuellen Batteriestand.

Queries und Cursor dienen der Kommunikation mit dem ContentProvider oder über den ContentProvider hinweg. Die bereitgestellten Operationen sind das Erstellen, das Abändern, die Anfrage sowie das Löschen von Einträgen. Dabei erfolgt die eigentliche Kommunikation von Queries und Cursor ebenfalls wieder auf der Grundlage der RPC-Semantik.

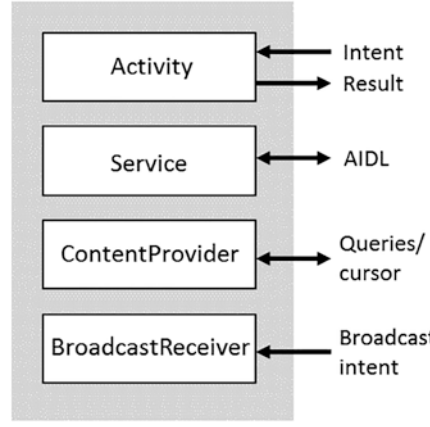

Abb. 7.3 Android-Kommunikationswege nach Komponenten. (Quelle: eigene Darstellung)

7.3 Signaturen und Zertifikate unter Android

7.3.1 Digitales Signieren von Android-Apps

Das digitale Signieren von Apps war unter Android von Beginn an vorgesehen. Im Verlauf der Zeit, insbesondere mit den Versionen Android 7 und Android 9, ist es allerdings noch einmal grundlegend überarbeitet worden. Dabei ist es das erklärte Ziel den Entwickler einer App durch die Nutzung von Verfahren der asymmetrischen Kryptografie nachweisen zu können. Der App-Entwickler erstellt hierzu selbst einen öffentlichen Schlüssel k_{pub} und einen zu seinem öffentlichen Schlüssel passenden privaten Schlüssel k_{pr}. Diesen verwendet er zum anschließenden digitalen Signieren des APK-Archivs. In seiner allgemeinen Form haben wir das digitale Signieren von Nachrichten mittels asymmetrischer Kryptografie bereits in Abschn. 2.4.4 des Grundlagenteils kennen gelernt. Im Falle des digitalen Signierens von APK-Archiven ist es allerdings ausdrücklich erlaubt, als Entwickler ein selbst-signiertes Zertifikat $Cert\left(ID_{Ent}, k_{pub}\right)$ zu erstellen und dieses dem Smartphone-Nutzer zur späteren Überprüfung des signierten APK-Archivs mitzuteilen. Denn die Android-Sicherheitsarchitektur sieht es für das Signieren von Apps nicht vor, eine zentrale, beiderseitig vertrauenswürdige Autorität wie eine Zertifizierungsstelle einzubinden. Dieser Sachverhalt ist in der Abb. 7.4 noch einmal dargestellt, wobei aus Gründen der Übersichtlichkeit das Übertragen des selbstsignierten Entwicklerzertifikates und des APK-Archivs mitsamt der Signatur $s$getrennt voneinander dargestellt sind.

Durch das digitale Signieren des APK-Archivs soll beispielsweise beim Einspielen von Updates für eine App sichergestellt werden, dass diese vom selben Entwickler stammen wie die ursprünglich installierte App. Man mache sich klar: Da die technische Umsetzung digitaler Signaturen insbesondere auch das Schutzziel der *Nicht-Abstreitbarkeit* erfüllen, kann der Entwickler jederzeit gegenüber einer dritten Instanz beweisen, der Urheber des signierten Archivs gewesen zu sein. Dies gilt allerdings

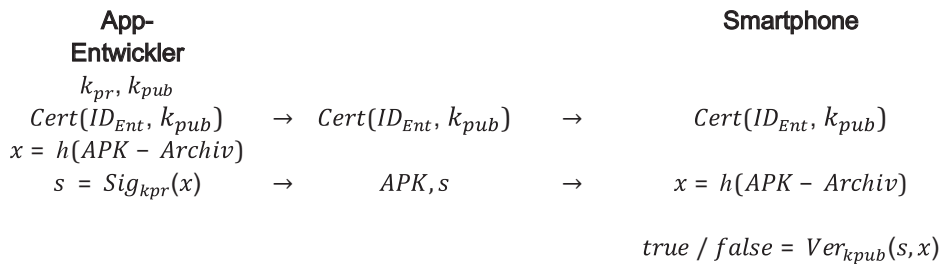

Abb. 7.4 Signieren von APK-Archiven unter Verwendung eines selbstsignierten Zertifikates $Cert\left(ID_{Ent}, k_{pub}\right)$ des Entwicklers. (Quelle: eigene Darstellung)

auch nur, wenn er plausibel nachweisen kann, den privaten Schlüssel k_{pr} sicher aufbewahrt zu haben. Inwieweit ein derartiger Nachweis dann tatsächlich als Beweis für die Rechtsprechung genügt und vor Gericht Bestand hätte, ist allerdings mehr als zweifelhaft. Neben anderen Aspekten dürfte ein selbstsigniertes Zertifikat $Cert\left(ID_{Ent}, k_{pub}\right)$, über das einzig der Entwickler bestätigt, dass seine ID an den verbreiteten öffentlichen Schlüssel gebunden ist, hierzu bei weitem nicht ausreichen.

7.3.2 Zertifikate unter Android

Android unterscheidet generell zwischen Nutzerzertifikaten und Systemzertifikaten. Allerdings haben ältere Android-Versionen vor der Android-Version 7 Nutzerzertifikate auf die gleiche Art und Weise wie Systemzertifikate behandelt. Erst ab der Version Android 7 wird Nutzerzertifikaten nicht mehr systemweit vertraut, auch wenn diese nach wie vor noch installierbar sind. Eine weitere Neuerung ab Android 7 ist die *Network Security Configuration.* Hierüber können Entwickler komfortabler Konfigurationen in Bezug auf ihre vertrauenswürdige Verwendung von Zertifikaten für eine gesicherte Netzwerk-Kommunikation durchführen. Neben anderen Konfigurationsmöglichkeiten kann der Entwickler nun Einstellungen zu *Vertrauensankern*, sowie dem *Zertifikats-Pinning* vornehmen. Mithilfe derartiger Vertrauensanker ist es dem Entwickler beispielsweise möglich, festzulegen ob selbstsignierte Zertifikate verwendet werden können. In Anlehnung an [8] würde dies in der `network_security_config.xml` folgendermaßen auszugestalten sein:

```
:
<network-security-config>
:
            <trust-anchors>
            <certificates src="@raw/extracas"/>
            <certificates src="system"/>
            </trust-anchors>
:
</network-security-config>
```

Ein Zertifikats-Pinning kann vorgenommen werden, indem in die Konfigurationsdatei diejenigen Zertifikate eingetragen werden, die für den Aufbau einer gesicherten SSL/TLS-Verbindung zu nutzen sind. So soll sichergestellt werden, dass unter der Sicherheitsannahme *„Trust-on-First-Use“* (ToFU) das Zertifikat an einen PIN gebunden ist. Dabei ist ein PIN nichts anderes als ein Hashwert beispielsweise unter Verwendung des Hashverfahrens SHA-256. Dieser geht dann mit jedem Kontakt zum Server wieder mit in die Anfrage ein. Die Sicherheitsannahme „Trust-on-First-Use“ besagt, dass unter der Annahme, dass beim erstmaligen Kontakt zwischen beiden Parteien kein Angreifer

Mallory zugegen ist, von da an die Kopplung der gesicherten Verbindung an das Zertifikat als gesichert gelten kann. Der entsprechende Eintrag hierzu würde wiederum in der `network_security_config.xml` zu erfolgen haben:

```
:
<network-security-config>
                    :
          <pin-set expiration="2020-01-01">
          <pin digest="SHA-256">7HI……RhJ3Y=</pin>
          <!—backup pin -->
          <pin digest="SHA-256">fwz……DM1oE=</pin>
          </pin-set>
                    :
</network-security-config>
```

Allerdings werden wir im weiteren Verlauf dieses Kapitels erfahren müssen, dass mittlerweile eine Reihe von Werkzeugen verfügbar sind, mit denen das oben vorgestellte Zertifikats-Pinning, auch geläufig unter dem Begriff Public Key Pinning, unterlaufen werden kann. Nachdem wir dedizierte Techniken zum Zertifikats-Pinning erörtert haben, werden wir in den Abschn. 9.1.3 und 9.1.4 erfahren, welche Techniken Sicherheitsexperten aufzeigen, um ein Zertifikats-Pinning ohne und mit Obfuskierung erfolgreich zu umgehen.

7.4 Das Rechtemodell von Android

Man muss es so deutlich sagen: Das Rechtemodell von Android ist eher verwirrend. Dies liegt im Wesentlichen daran, dass es unterschiedliche Rechtemodelle auf verschiedenen Hierarchiestufen bündelt, was für deren Übersichtlichkeit und eine zweifelsfreie Handhabbarkeit der Rechtevergabe auch für geübte Nutzer und Entwickler nicht eben vorteilhaft ist.

Auf der untersten Ebene wird die Einhaltung vergebener Rechte auf der Basis von Benutzer-IDs (UID) und Gruppen-IDs (GID) von Datei- und Ausführungsrechten durch den Betriebssystemkern überwacht. Des Weiteren laufen Endnutzer-Anwendungen, also die Apps, als dedizierte Prozesse innerhalb der ART. Mit der Installation auf dem mobilen Gerät erhält eine App eine feste UID. Diese ist entwicklerspezifisch, das heißt, dass jede App, die mit dem gleichen Entwicklerschlüssel signiert worden ist, auf dem Smartphone auch die gleiche UID erhält. Darüber hinaus gibt es auf der obersten Ebene des Rechtemodells Android-spezifische Rechte. Unter Android 2.2 wurden 134 Rechte in drei übergeordnete Kategorien eingeordnet. Dabei fasst die Kategorie *normal permissions* solche Rechte zusammen, welche zur Erbringung der Funktionalität ‚ohne direkte Risiken' notwendig sind. Hierunter fallen beispielsweise BLUETOOTH, BLUETOOTH_ADMIN oder NFC. Dabei werden wir in Abschn. 7.6 noch genauer erörtern,

dass auch die Vergabe solcher NFC-Rechte ‚ohne direkte Risiken' sehr wohl ein nicht zu unterschätzendes Schadenspotenzial mit sich bringen kann.

Aber auch die Andoid-Rechtekategorie *zero-permission* ist es wert ein wenig genauer betrachtet zu werden. So sind jüngst einige Sicherheitsforscher der Frage nachgegangen, was eigentlich im Detail die Einstellung *zero-permission* für eine App unter Android bewirkt. Dabei haben sie untersucht, welche Zugriffsrechte ein Smartphone, ausgestattet mit Android Pie, einer App dennoch einräumt [9]. Sie kamen zu recht bemerkenswerten Ergebnissen, indem sie eine von ihnen entwickelte App namens *Ferret Profiler* getestet haben. Diese App war in der Lage, Daten aus acht Kategorien zu sammeln (siehe Tab. 7.1)

Die Kategorie *dangerous permissions* bündelt all diejenigen Rechte, die für den Zugriff auf private Dateien sowie sensible Funktionalitäten erforderlich sind. Diese sind vom Nutzer des Smartphones zu bestätigen, andernfalls kann die App nicht auf diese Ressourcen zugreifen. Die Kategorie *Signature/System permissions* schließlich beinhaltet diejenigen Berechtigungen einer App, die von allen Apps zu nutzen sind, die mit dem gleichen Entwicklerschlüssel signiert worden sind wie die zu installierende App. Generell werden die von einer App angeforderten Erlaubnisse in deren Manifest-Datei über das *Tag* `<uses-permission>` festgelegt. Mit dem Android Asset Packaging Tool lassen sich neben anderen Dingen auch die Berechtigungen einer App anzeigen:

```
$: aapt d permissions "meineapp.apk"
```

Die Rückgabe wäre dann beispielsweise

```
package: com.app.meineapp
uses-permission: android.permission.ACCESS_NETWORK_STATE
uses-permission: android.permission.INTERNET
```

Tab. 7.1 Der App *Ferret Profiler* zugestandene Rechte trotz der Vergabe der Rechtekategorie *zero-permission.* (Quelle:[9])

K1	Gerätehersteller, NFC-Fähigkeit, verfügbare Kameras, Gerätesprache, Kompass, Android-Version,…
K2	DeviceLocked, DeviceSecure, Rooted Device, Keyguard Secure
K3	Netzwerkanbieter, Netzwerktyp, Netzwerkland, Roaming und Zustand des GSM-Anrufs
K4	VoIP-Anrufzustand
K5	Alarm-Uhrzeit
K6	Liste der installierten Apps
K7	Daten auf dem Clipboard
K8	Zustand der Kamera

Der ausschlaggebende Punkt hinsichtlich des Android-Rechtemodells ist aber wohl folgender: Der Nutzer hat bei der Installation einer App zu bestätigen, dass er den angefragten Rechten seine Zustimmung erteilt. Neben der Offenkundigkeit, dass sich hierbei oftmals ein Zielkonflikt zwischen der erhofften Nutzung der App und der Bewilligung der angefragten Rechte ergibt, delegiert Google damit die Problematik der Rechtevergabe und die daraus resultierenden Sicherheits- und Datenschutzfragen an den oftmals allzu unbedarften Nutzer. Woher soll dieser auch wissen, dass die (bewusst) irreführende Bezeichnung *zero-permission* einer App sehr wohl noch den Zugriff auf die oben genannten acht Kategorien gewährt?

7.4.1 Zusammenwirken von Android mit SELinux

Das ursprüngliche Sicherheitsmodell von Android überwacht den Zugriff der einzelnen App-Komponenten sowie der Systemressourcen auf der Anwendungsebene. Darüber hinaus bietet es Sandboxing und Isolierung auf der Kernel-Ebene. Hierzu verwendet Android die klassische Zugangskontrolle *Discretionary Access Control* (DAC), um Apps gegenseitig voneinander sowie gegenüber dem eigentlichen System zu isolieren. DAC basiert auf UIDs und GIDs und bietet somit eine doch recht grobgranulare Zugriffskontrolle für die Prozesse einzelner Apps. Jede App erhält bereits bei ihrer Installation eine eindeutige UID und GID. Dabei ist die Vergabe einer UID entwicklerspezifisch. Dies bedeutet, dass alle Anwendungen des gleichen Entwicklers, die mit dem gleichen Entwicklerschlüssel signiert worden sind, die gleiche UID erhalten.

Security Enhanced Linux (SELinux) ist eine Erweiterung des Linux-Kernels, die ab der Version Linux 2.6x bereits im Kernel enthalten ist. Die Entwicklung wurde ursprünglich von der National Security Agency (NSA) initiiert. Mit der Zeit haben dann mehr und mehr Unternehmen zur Entwicklung beigetragen. Ab der Version 4.3 ist SELinux auch Bestandteil des Android-Betriebssystems. Neben weiteren Aspekten, auf die wir hier nicht weiter eingehen möchten, unterstützt es die Durchsetzung von Zugriffskontrollen auf Ressourcen auf der Basis von *Mandatory Access Control* (MAC) [10, S. 209]. Dieses Sicherheitsmodell ist damit über die klassische *Discretionary Access Control* (DAC) [10, S. 293] hinaus umgesetzt und mit dem Einsatz von SELinux verfügbar. Mandatory Access Control ermöglicht es, systemweite Richtlinien für Prozesse, Objekte und Operationen auf der Grundlage von Informationsflüssen durchzusetzen. Anders als bei anderen Sicherheitsmodellen für IT-Systeme gehen bei einer Umsetzung des Sicherheitsmodells Mandatory Access Control neben dem Subjekt – der jeweilige Nutzer – und dem Objekt – der Ressource, auf die zuzugreifen ist – auch Regeln und Eigenschaften ein, inwieweit ein Subjekt ein Objekt nutzen darf. So erlaubt die technische Umsetzung eines Sicherheitsmodells wie Mandatory Access Control auf Kernel-Ebene prinzipiell eine sehr fein granulare Umsetzung von Integritäts- und Vertraulichkeitsschutz unterschiedlichster Ressourcen, wie beispielsweise der von Dateien, Verzeichnissen oder dem Netzzugang. Dies geschieht folgendermaßen:

1. Objekte erhalten eine Sicherheitsmarkierung (engl. *label*), die in erweiterten Attributen zum jeweiligen Objekt durch Linux abgelegt werden (Nutzer: Rolle: Typ/Domain).
2. Auch Subjekte (Prozesse) erhalten Sicherheitsmarkierungen.

Dabei gibt eine Sicherheitsmarkierung, die einem zu kontrollierenden Objekt zugewiesen ist, Auskunft über seinen Grad der *Sensitivität.* Eine Sicherheitsmarkierung, die einem registrierten Subjekt zugewiesen worden ist, gibt hingegen Auskunft über dessen *Vertrauenswürdigkeit.* Diese wird über eine Freigabemarkierung gesetzt.

Die Schwierigkeit bei einer auf Mandatory Access Control basierenden Zugriffskontrolle besteht nun darin, die Sicherheitsmarkierung für alle Objekte und Subjekte entsprechend der Sicherheitsrichtlinie korrekt umzusetzen und anzupassen. Hierbei ist es wichtig, sich die Richtung möglicher Informationsflüsse zu vergegenwärtigen: Denn alle *lesenden* Operationen erzeugen einen Informationsfluss weg vom Objekt, auf welches zugegriffen wird, hin zu dem registrierten Subjekt. Alle schreibenden Operationen hingegen erzeugen einen Informationsfluss genau in die umgekehrte Richtung, also weg vom Subjekt und hin zum Objekt auf das zugegriffen werden soll. Für beide Operationsarten wird die Flussrichtung auf der Basis von *Mandatory Access Control* anhand der Relation „weniger vertrauenswürdig“ (im nachfolgenden Pseudocode notiert mit <=) auf den vergebenen Sicherheitslabeln `l_subject` und `l_object` von Subjekten `subject` und Objekten `object` gewährt oder eben unterbunden. Der Pseudocode für eine Regelung der systeminternen Informationsflüsse könnte folgendermaßen aussehen:

```
if(operation_mode == write)
   allow information_flow if l_subject <= l_object;
else if(operation_mode == read)
   allow information_flow if l_object <= l_subject;
else
   deny information_flow;
```

SELinux kann nicht gegen dedizierte Verwundbarkeiten des Kernels schützen und bietet auch nur dann einen erhöhten Schutz, wenn die Sicherheitsrichtlinien korrekt anhand der oben skizzierten Sicherheitsmarkierungen für Objekte und Ressourcen abgebildet wurden. Trotzdem lässt sich festhalten, dass es innerhalb des Betriebssystems Android genutzt werden kann, um Apps strikt voneinander zu separieren und eine horizontale Rechteausweitung (siehe Abschn. 7.5) zu unterbinden. Innerhalb von SELinux überwacht dabei eine Kernel-interne Komponente, der Security-Server, kontinuierlich anhand der Sicherheitsmarkierungen, welches Subjekt auf ein Objekt wie zugreifen darf. Entscheidend ist, dass dies nach dem *Least-Privilege-Prinzip* erfolgt. Dies bedeutet, dass nur wenn auch tatsächlich eine Richtlinie existiert, der Zugriff auf bzw. der Informationsfluss hin zu der jeweiligen Ressource gestattet wird. In allen anderen Fällen, und zwar unabhängig von den weiteren Rechten, die Linux diesem Objekt zugestanden hat, wird der Zugriff strikt untersagt.

Doch wie so häufig steckt bei einer praxistauglichen Umsetzung von Modellen jeglicher Art der Teufel im Detail. Dies gilt für SELinux insbesondere für Systemaufrufe wie ioctl(), über die Apps des *User-Space* die Möglichkeit erhalten, Zugriff auf Funktionen des *Kernel-Space* zu erlangen [11]. Eine feingranulare Umsetzung in Form von Sicherheitsmarkierungen ist hierbei nicht trivial.

Auf einem Android-Smartphone kann SELinux in verschiedenen Zuständen vorliegen: SELinux kann deaktiviert sein, es kann erzwungen sein oder aber als erlaubt (permissiv) eingestellt sein. Ist es im *deaktivierten* Zustand, so bietet SELinux keinerlei zusätzliche Sicherheit und es greifen ausschließlich die mittels Discretionary Access Control festgelegten Zugriffskontrollen. Wird es im Zustand *permissiv* betrieben, so ist der Sicherheitsgewinn ebenfalls denkbar gering. Denn nun können die Prozesse bösartiger Apps unter der UID des jeweiligen Entwicklers ebenfalls noch immer auf jegliche Ressourcen lesend, schreibend oder ausführend zugreifen. Der Unterschied zur Deaktivierung von SELinux besteht darin, dass nun die Mandatory-Access-Control-Richtlinien gelten, wodurch die App zwar nicht an ihrer Ausführung bzw. am Zugriff auf zu schützende Ressourcen gehindert wird, aber es wird zumindest protokolliert, wenn ein Zugriff auf derartige Ressourcen erfolgt ist. Allein im Zustand *erzwungen* verhindern die gesetzten Regeln zur Ausführungszeit einer App den Zugriff auf mittels Mandatory Access Control geschützter Ressourcen.

7.4.2 Android Keystore

Der Andoid Keystore [12] ist ein weiterer wesentlicher Baustein der Android-Sicherheitsarchitektur. Der Keystore ermöglicht das Erstellen, Speichern und Nutzen von kryptografisch sicherem Schlüsselmaterial in einer kontrollierten Umgebung. Was dies im Einzelnen bedeutet, hängt maßgeblich von dem Smartphone ab, auf welchem Android installiert ist. Denn der Android Keystore selbst ist erst einmal nur eine API des Android-Frameworks mit welchem dem Entwickler einer Anwendung Zugriff auf Keystore-Funktionalitäten ermöglicht wird. In Abhängigkeit von der konkreten Version des zum Einsatz kommenden Smartphones werden die erzeugten Schlüssel dann entweder rein softwareseitig durch das Betriebssystem selbst oder, was hinsichtlich der gebotenen Sicherheit als deutlich besser zu bewerten ist, mit Hilfe von hardwaregestützten Schutzmaßnahmen sicher abgelegt. So oder so: Mit dem Booten des Android-Gerätes wird der Keystore-Dienst aktiviert. Nun kann der Anwendungsprozess einer App, sofern seine Berechtigungen dies erlauben, Operationen auf den im Keystore abgelegten Schlüsseln einleiten, indem er einem für den Keystore verantwortlichen Systemprozess anweist, diese Operationen durchzuführen. Sofern hardwaregestützte Schutzmaßnahmen wie ein *Trusted Execution Environment* (TEE) oder *Secure Element* (SE) auf dem Gerät zur Verfügung stehen und darüber hinaus auch aktiviert sind, ist das Schlüsselmaterial niemals außerhalb der sicheren Hardware verfügbar.

Ein Trusted Execution Environment ist ein sicherer Bereich im Hauptspeicher, ausgestattet mit einem eigenen Betriebssystem. Kommuniziert wird mit dem Android-Betriebssystem über eine restriktive Schnittstelle. Zur Kommunikation dient ein gemeinsamer Speicherbereich, über den Daten ausgetauscht werden können. Ab der Android-Version 9 können die Schlüssel nun in einem Secure Element abgelegt werden. Dies ist ein eigener Mikrochip, der über eine eigene CPU, sicheren Speicher und dergleichen verfügt und darüber hinaus gegenüber einer Reihe von Seitenkanalangriffen gehärtet ist, auf die wir an dieser Stelle jedoch nicht weiter eingehen wollen. Um für eine Anwendung nachvollziehbar zu machen, wo genau ein Schlüsselpaar erstellt worden ist, wurde mit der Version Android 7 die sogenannte Key-Attestation eingeführt. Mit der Generierung der Schlüssel wird ein Attestierungszertifikat erstellt, aus dem hervorgeht, ob die Erstellung innerhalb eines Trusted Execution Environment oder aber einem Secure Element vollzogen wurde [13]. Außerdem ermöglicht eine Erweiterung ab Android 8 einer Anwendung mithilfe der sogenannten ID-Attestierung, die Hardware-ID eines Gerätes festzustellen [12].

Der Einsatz dieser Komponenten erhöht zweifelsfrei das Vertrauen in die Sicherheitsarchitektur von Android. Allerdings sollte man auch nicht verschweigen, dass es noch mindestens zwei weitere, für den Praxiseinsatz nicht ganz unwesentliche Hürden zu meistern gilt: Zum einen können hardwaregestützte Schutzmaßnahmen nur dann aktiviert werden, wenn die sichere Hardware auch die geforderten Kombinationen aus kryptografischen Bausteinen unterstützt, also die eingesetzten Verschlüsselungsalgorithmen, verwendete Blockmodi und dergleichen mehr. Zum anderen wäre es wünschenswert, eine tatsächliche Ende-zu-Ende-Vertraulichkeit zu gewährleisten. Konkret bedeutet dies, dass die Daten bis hin zur kryptografischen Hardware verschlüsselt sind, sodass selbst ein Mitlesen bei einem physischen Zugriff auf das Smartphone nicht möglich wäre. Letzteres ist jedoch bisher nicht gewährleistet und deren Umsetzung unter Umständen auch gar nicht angestrebt.

7.5 Rechteausweitung unter Android

Für gewöhnlich lassen sich zwei Arten der Rechteausweitung (engl. *privilege escalation*) unterscheiden: die *vertikale* Rechteausweitung und die *horizontale* Rechteausweitung. Man spricht von einer vertikalen Rechteausweitung, wenn aufgrund von Programmierfehlern oder einem lückenhaft konzipierten Entwurf ein Benutzer mehr Funktionalitäten des Systems erlangen kann, als dies ursprünglich vorgesehen war. Hierzu sind normalerweise weitere oder höhere Ausführungsrechte auf den verschiedenen Ressourcen des Systems erforderlich. Hingegen spricht man von einer horizontalen Rechteausweitung immer dann, wenn verschiedene Benutzer der gleichen Ebene Funktionen und Daten anderer Benutzer erhalten, ohne dass dies ursprünglich so intendiert gewesen wäre. Die horizontale Rechteausweitung soll uns nachfolgend noch ein wenig genauer beschäftigen. Man kann sich leicht vorstellen, dass eine derartige horizontale Rechteausweitung unter

Android durch die im Vorfeld beschriebenen Kommunikationsmöglichkeiten mittels Interprozesskommunikation (IPC) zwischen App-Komponenten stark begünstigt wird. Denn dies ist immer genau dann gegeben, wenn die Kommunikation zwischen Apps in einer unkontrollierten und vorab nicht beabsichtigten Art und Weise erfolgt.

Man stelle sich beispielsweise die beiden Apps TaskScheduler und SMS-Formatter vor. Während die erste App aufgrund des ihr zugestandenen Zugriffsrechtes android.permission.INTERNET Daten ins Internet senden und von dort empfangen kann, ist dies der App SMS-Formatter untersagt. Diese hat aufgrund des ihr zugestandenen Zugriffsrechtes android.permission.READ_SMS allerdings die Möglichkeit eine SMS aus der lokalen Datenbank zu lesen oder dorthin zu schreiben. Würde man also die Rechte der Apps über folgende Aufrufe aus der Shell

```
$: aapt d permissions "TaskScheduler.apk"
```

sowie

```
$: aapt d permissions "SMS-Formatter.apk"
```

in Erfahrung bringen wollen, so würde im ersten Fall

```
uses-permission: name='android.permission.INTERNET'
```

und im zweiten Fall

```
uses-permission: name='android.permission.READ_SMS'
```

zur Anzeige kommen. Da darüber hinaus diese beiden Apps jedoch mit AIDL, Broadcast Intents oder dergleichen über IPC kommunizieren könnten, besteht jedoch sehr wohl die Möglichkeit auch für die App SMS-Formatter unter Einbindung des TaskScheduler die Berechtigung für das Senden von Daten ins Internet zu ergaunern. Dieser Sachverhalt ist in der Abb. 7.5 in angepasster Form aus [14] übernommen.

Solch eine horizontale Rechteausweitung lässt sich weiter unterscheiden in solche, bei der eine bösartige App eine ‚irritierte aber gutartige' Partei in Form einer App (engl. *confused deputy*) einbindet, und solche, bei der zwei bösartig agierende Apps, die aufeinander zugeschnitten sind, ihre Berechtigungen erweitern. Der prinzipielle Unterschied dieser beiden Formen einer horizontalen Rechteausweitung zwischen Apps ist in der Abb. 7.6 noch einmal veranschaulicht.

7.5.1 Ansätze zur Abschwächung horizontaler Rechteausweitung

In den letzten Jahren haben sich eine ganze Reihe von Forschergruppen damit beschäftig technische Maßnahmen zur Eindämmung oder zumindest der Erkennbarkeit der oben

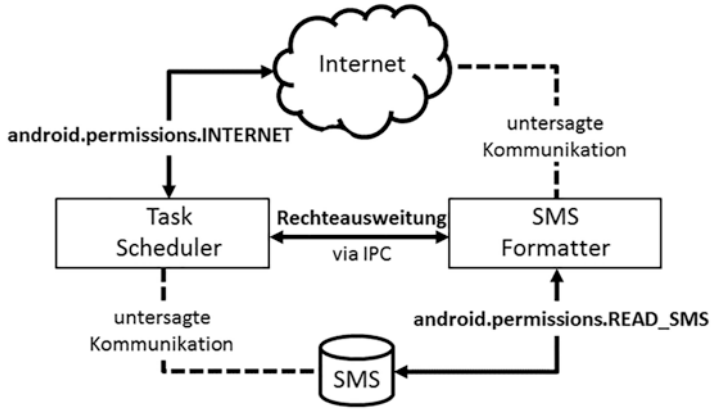

Abb. 7.5 Horizontale Rechteausweitung aufgrund von IPC-Nutzung zwischen den Apps SMS-Formatter und Task-Scheduler. (Quelle: eigene Darstellung in Anlehnung an [14])

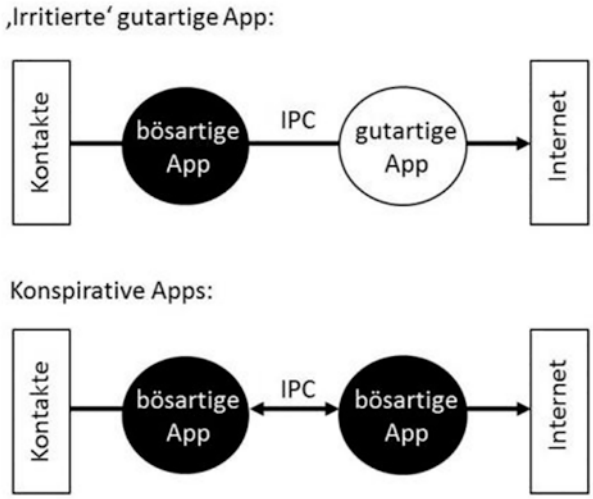

Abb. 7.6 Varianten der horizontalen Rechteausweitung durch IPC-Nutzung: IPC zwischen bösartiger App und gutartiger App oder konspirative IPC zwischen zwei bösartigen Apps. (Quelle: eigene Darstellung in Anlehnung an [14])

geschilderten Formen einer horizontalen Rechteausweitung durch IPC zu entwickeln und umzusetzen. Zu nennen sind hier neben anderen der Ansatz TaintDroid [15] aus dem Jahr 2010, die Ansätze XmanDroid und IPC Inspection aus dem Jahre 2011 sowie QuantDroid [14] und QuantDroid++ [16]. Sowohl QuantDroid als auch QuantDroid++ setzen auf TaintDroid auf und sollen hier stellvertretend für andere Ansätze erörtert werden. Während sich QuantDroid mit Möglichkeiten der Abschwächung oder Erkennbarkeit horizontaler Rechteausweitung für *flüchtige* IPC-Kommunikation zwischen Apps auseinandersetzt, zielt der Ansatz QuantDroid++ auf die Reduzierung bzw. Erkennbarkeit horizontaler Rechteausweitung für *persistente* IPC-Kommunikation ab, also solche,

bei der die Daten über die Lebenszeit der schreibenden Apps hinaus gespeichert werden. Dies ermöglicht einer weiteren App, die Daten zu einem deutlich späteren Zeitpunkt von dem persistenten Speicher abzurufen, sodass zwischen diesen beiden konspirativen Apps erst einmal keinerlei zeitlicher Bezug feststellbar ist.

7.5.1.1 Abschwächung horizontaler Rechteausweitung bei synchroner Inter-App- Kommunikation

Zur Abschwächung der horizontalen Rechteausweitung bei einer flüchtigen Kommunikation zwischen Apps verwendet der Ansatz QuantDroid eine Komponente namens FlowGraph. Der FlowGraph-Dienst überwacht die Kommunikation zwischen den aktuell instanziierten DVMs und setzt hierzu auf TaintDroid auf. Die FlowGraph-Schnittstellendefinition IFlowGraph erfolgt wie gehabt in AIDL und enthält Schnittstellen für Methoden wie *spwanProcess, exitProcess, preCommunication* und *currentGraphState*. Dabei wird als Kommunikation wiederum Binder-IPC mit RPC-Client- und Server-Stub verwendet. Diese Erweiterung fängt Nachrichten im Server-Stub ab und leitet eine Reihe von Kommunikationsparametern an den FlowGraphDienst. Im Einzelnen sind dies die Kommunikationsparameter PID des Initiators, UID des Initiators, PID des Empfängers, UID des Empfängers, Größe der empfangenen Nachricht in Bytes, sowie das Markierungslabel. Die QuantDroid-Erweiterung zum Überwachen der Kommunikation mittels Intents erfordert Modifizierungen innerhalb der High-Level-Middleware des Binder-Frameworks, namentlich innerhalb des *ActivityManagerService* und des *ActivityStack*. Durch Aufruf von *preCommunication* erhält der FlowGraph-Dienst nun die oben genannten Kommunikationsparameter. Ähnliche Erweiterungen sind für eine nachvollziehbare Aufbereitung der flüchtigen Kommunikation zwischen Apps mit Hilfe von BroadcastIntents erforderlich. BroadcastIntents sind unter Android in der Klasse ActivityManagerService umgesetzt und dort in der Methode processCurrentBroadcastLocked(). Das Aussortieren erfolgt nun im IntentFilter der Manifest-Datei derjenigen App, die einen BroadcastIntent absetzt. Auch hier erfolgt die Übermittlung der Kommunikationsparameter an den FlowGraph-Dienst mittels der Methode preCommunication.

Der QuantDroid-Ansatz basiert auf der Vergabe von Schwellenwerten. So kann bei konspirierenden böswilligen Apps, die als unterschiedliche Systemnutzer laufen und darauf abzielen, Kontakte abfließen zu lassen, die IPC zwischen diesen Apps ab einem Schwellenwert von beispielsweise 1000 Bytes/min unterbrochen werden, indem der FlowGraph Dienst die sendende App hart terminiert. Ähnliches gilt für den Missbrauch von Schnittstellen.

▶ Taint-basierte Ansätze zur Abschwächung horizontaler Rechteausweitung gegen zeitgleiche Inter-App-Kommunikation stoßen immer dann an ihre Grenzen, wenn i) die Schwellenwerte der zu übertragenden Daten nicht sinnvoll konfiguriert wurden oder aber ii) die Verfahren synchroner IPC-Kommunikation umgangen werden und eine zeitlich versetzte asynchrone IPC-Kommunikation erfolgt. Die Verwundbarkeit resultiert aus dem **Ausnutzen einer unvollständigen Konzeptionierung.**

7.5.1.2 Abschwächung horizontaler Rechteausweitung bei asynchroner Inter-App-Kommunikation

Der oben aufgezeigte QuantDroid-Ansatz funktioniert zur Abschwächung der horizontalen Rechteausweitung bei quasi zeitgleicher synchroner Inter-App-Kommunikation mit allen Nachteilen, die ein Schwellenwert-basierter Ansatz zweifelsohne mit sich bringt. Leider versagt der Ansatz genau dann, wenn die Kommunikation zwischen Apps zeitlich versetzt asynchron erfolgt. Denn in diesem Fall werden die Daten von einer App persistent in eine lokale Datenbank geschrieben und von einer weiteren App später, möglicherweise lange nachdem die erste App beendet wurde, ausgelesen. Doch dieser Sachverhalt ist für die Erkennung einer horizontalen Rechteausweitung mit dem des Ansatz QuantDroid problematisch. Denn dieser funktioniert nur, da die Daten von unterschiedlichen Quellen mittels Taints (Fleck) markiert sind [15]. Schreibt man die Daten jedoch in eine Datenbank, so gehen diese Taints verloren und die Information einer Verbindung via Datenbank kann über den FlowChart-Dienst sowie ähnliche Ansätze zwangsläufig nicht mehr nachvollzogen werden. Dieser Sachverhalt ist in Anlehnung an [16] in Abb. 7.7 aufgezeigt. Auf der Übertragungsstrecke zwischen SMS-Datenbank, der SMS-Formatter-App sowie der persistenten Datenablage wird der Taint ‚SMS' erstellt und abgelegt. Allerdings ist er bei einem späteren Auslesen aus der Datenbank auf der Übertragungsstrecke zwischen Datenbank, Task-Scheduler und Schnittstelle zur Internetkommunikation nicht mehr sichtbar und kann daher zur Erkennung einer horizontalen Rechteausweitung nicht mehr verwendet werden.

Die grundlegende Idee, die Katharina Mollus nun verfolgte, besteht darin, solche Taints auch bei der Ablage in einer Datenbank zu erhalten. Dazu hat sie innerhalb der SQLiteDatabase-Klasse, die Android zur persistenten Datenablage nutzt, die Methoden insert(), query(), update() und delete() angepasst und auf diese Weise sichergestellt, dass die Taints ebenfalls persistent in einer eigens für alle Taints angelegten zentralen Datenbank gehalten werden. Werden also aufgrund asynchroner IPC-Kommunikation Daten von einer App persistent in der hierfür vorgesehenen Datenbank abgelegt, so werden mit

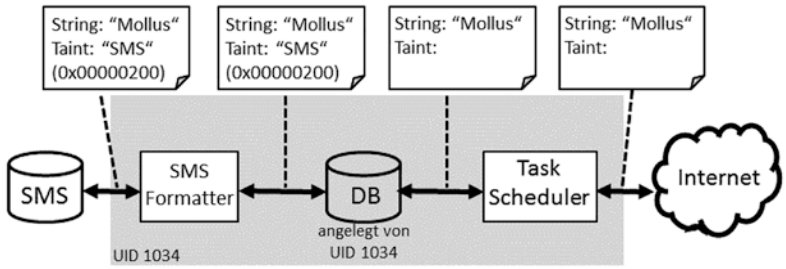

Abb. 7.7 Das Versagen Taint-basierter Ansätze zur Erkennung horizontaler Rechteausweitung bei zeitversetzter asynchroner Kommunikation und Ablage in einer Datenbank. (Quelle: eigene Darstellung in Anlehnung an [16])

QuantDroid++ deren Taints nun ebenfalls in der zentralen Taint-Datenbank gespeichert. Werden dann zu einem späteren Zeitpunkt die Daten von einer anderen App angefragt, so werden ebenfalls die korrespondierenden Taints aus der zentralen Taint-Datenbank gelesen. Auf diese Weise ist es dem QuantDroid-FlowGraph-Dienst nun möglich, auch eine zeitlich versetzte Kommunikation zwischen zwei Apps zu überwachen und bei der Überschreitung eines Schwellenwertes bedarfsweise den sendenden App-Prozess zu terminieren. Allerdings stößt der QuantDroid++-Ansatz bei der technischen Umsetzung der Fragestellung, wie lange Taints sinnvollerweise persistent vorgehalten werden können, an seine Grenzen. Eigentlich kann mit diesem Ansatz auch eine Rechteausweitung zwischen konspirativen Apps erkannt werden deren Aktivierungen durch den Nutzer Wochen auseinanderliegen. Allerdings wird die zentrale Taint-Datenbank auf dem Smartphone schon deutlich früher zu groß geworden sein, sodass die Erkennung der Rechteausweitung über asynchrone IPC-Kommunikation immer an ein fixes Zeitfenster gekoppelt ist da die Datenbank von Zeit zu Zeit zumindest teilweise gelöscht werden muss.

- Taint-basierte Ansätze zur Abschwächung horizontaler Rechteausweitung auch gegenüber einer zeitlich versetzten Kommunikation von Apps stoßen immer dann an ihre Grenzen, wenn i) der Schwellenwert nicht sinnvoll konfigurierte wurde oder aber ii) eine asynchrone Kommunikation zwischen konspirierenden Apps in einem zu langen Zeitabstand erfolgt. Dann wächst die Taint-Datenbank zu stark an und ältere persistente IPC-Vorgänge können nicht mehr erkannt werden. Die Verwundbarkeit resultiert aus dem **Ausnutzen einer unvollständigen Konzeptionierung** sowie **Skalierungsproblemen bei der Implementierung**.

7.6 Android und NFC

Im ersten Teil dieses Buches haben wir uns schon einmal mit der Near Field Communication (NFC) als einer eigenständigen Kommunikationstechnologie beschäftigt und Verwundbarkeiten, Schwachstellen sowie Angriffe auf verschiedene Tag-Typen aufgezeigt. In diesem Abschnitt betrachten wir die NFC-Technologie aus einem etwas anderen Blickwinkel. Nun ist es unser Anliegen, NFC im Zusammenspiel mit modernen mobilen Betriebssystemen wie Android oder iOS zu erörtern. So unterstützen Android-Smartphones den NFC-Standard und nehmen Nachrichten im *NFC Data Exchange Format* (NDEF) entgegen. Android bietet gleich mehrere NDEF-Funktionen für Visitenkarten, das Öffnen von Links, WLAN- bzw. Bluetooth-Verbindungen, das Starten von Apps oder das Senden einer SMS und noch einiges mehr. Wie steht es also mit der Verwundbarkeit eines Smartphones, das sich in der Nähe eines NFC-Tags befindet und NFC aktiviert hat? Da bei Android die Aktivierung von NFC die Voreinstellung ist, dürfte es aktuell genügend Smartphones geben, die in der Nähe eines NFC-Tags dessen NDEF-Nachrichten entgegennehmen und versuchen, diese zu interpretieren. Wenn Mallory also ein Tag so beschreibt, dass eine

von diesem präparierten Tag gesendete NDEF-Nachricht Aktivitäten auf dem Smartphone anstößt, die bösartig sind oder aber das Sicherheitsniveau des Gerätes heruntersetzen, dann wäre dies eine sehr ernsthafte Sicherheitslücke. Dieser Frage ist Jan Breig im Rahmen seiner Bachelorarbeit nachgegangen, zumal kostenlose Android-Apps verfügbar sind, mit denen die angestrebte Funktionalität einfach auf ein Tag geschrieben werden kann. Es kann also grundsätzlich nicht ausgeschlossen werden, das über diesen Kanal Angriffe auf ein Smartphone durchgeführt oder zumindest vorbereitet werden. Dieses Phänomen ist nicht ganz neu. Roel Verdult und François Kooman [17] haben bereits im Jahre 2011 bei einem NFC-fähigen Mobiltelefon *Nokia 6212 Classic* gezeigt, dass beim Scannen eines schädlichen RFID-Tags das Mobiltelefon sich mit einem Bluetooth-fähigen Gerät in der Nähe verbindet und mit diesem Daten sendet und empfängt.

Durch Untersuchung der NFC-Kommunikation mit dem Werkzeug Proxmark3 hat der Sicherheitsexperte Jan Breig bei einem Smartphone mit Android 9 festgestellt, dass bei eingeschaltetem Display das elektromagnetische Feld aktiviert ist. Bei einem gesperrten Display werden die eingehenden NDEF-Nachrichten allerdings nicht ausgewertet. Eine Auswertung findet nur bei angeschaltetem Smartphone und entsperrtem Display statt. Durch Ansprechen der Funktion *Link öffnen* mittels einer NDEF-Nachricht ist es in diesem Zustand eines Android-9-Smartphones beispielsweise möglich, ohne die explizite Einwilligung des Nutzers eine Webseite im Browser des Smartphones zu öffnen. Ein Angreifer könnte somit auf diese Weise die Kontrolle über das Smartphone übernehmen, wenn er auf der so geladenen Webseite gezielt einen Android -Exploit platziert hat. Da der Nutzer beim Laden der Seite keinerlei Bestätigung durchführen muss, ist er damit selbst nicht in der Lage, einen derartigen Angriff zu unterbinden. Dieses Verhalten unterscheidet sich von dem anderer NDEF-Funktionen, bei denen der Nutzer jeweils durch eine Bestätigungsanfrage eingebunden ist. So ist der unerkannte Aufbau einer WLAN- oder Bluetooth-Verbindung mittels präparierten NDEF-Nachrichten, ausgelöst von einem Tag in der Nähe. eher schwierig unerkannt durchzuführen. Generell lässt sich aber sicherlich feststellen, dass es als durchaus heikel zu bewerten ist, wenn NFC auf dem Smartphone permanent aktiviert ist.

7.7 Zusammenfassung

Wir sind in den Themenbereich zu Softwarekomponenten mobiler digitaler Geräte eingestiegen indem wir einen Blick unter die Motorhaube des mobilen Betriebssystems Android getätigt haben. Dabei haben wir uns mit den Grundlagen der Android-Systemarchitektur vertraut gemacht und anschließend die Abfolge des Bootvorgangs erörtert. Hierbei haben uns insbesondere der Zygote-Prozess, sowie die ART interessiert. Neuere Versionen von Android unterstützen das abgesicherte Hochfahren, bei dem eine Vertrauenskette beim Starten des Gerätes aufgebaut wird mit dem Ziel, die Integrität jeder genutzten Ressource zu prüfen und im Bedarfsfall den Bootvorgang

abzubrechen. Anschließend haben wir, nachdem wir die Komponenten einer App erörtert haben, mögliche Kommunikationswege zwischen Android-Applikationen besprochen und dabei Parallelen zu einem verteilten System gezogen. Danach haben wir erörtert, wie der Entwickler einer App über das Anfügen einer digitalen Signatur über das gesamte APK-Archiv den Entwickler einer App kenntlich macht. Dies geschieht allerdings unter Verwendung eines selbstsignierten Entwicklerzertifikates. Generell haben wir besprochen, dass Android zwischen Nutzerzertifikaten und Systemzertifikaten unterscheidet. Auch wenn Nutzerzertifikate nach wie vor installierbar sind, so wird diesen ab der Version 7 nicht mehr systemweit vertraut. Über die Network Security Configuration kann der Entwickler nun Vertrauensanker für Zertifikate setzen, sowie das Zertifikats-Pinning vornehmen. Bei der Erörterung des Android-Rechtemodells fällt auf, dass dieses doch recht verwirrend ist, da es hierbei verschiedene Hierarchiestufen gibt. Allerdings ist auch die Kategorisierung der einer App zugestandenen Zugriffsrechte teilweise irreführend. Apps, denen beispielsweise die Rechtekategorie *zero-permission* zugewiesen wurde, haben dennoch die Möglichkeit, auf einzelne Ressourcen zuzugreifen oder aber deren Zustand abzufragen. Eine echte konzeptionelle Besserung, was die Ausgestaltung der Zugriffsrechte angeht, verspricht die Verwendung von *Mandatory Access Control* durch SELinux. Allerdings muss es auch tatsächlich verwendet werden, was bedeutet, dass Sicherheitsmarkierungen vergeben werden müssen und SELinux auch wirklich aktiviert worden ist. Darüber hinaus haben wir den Android Keystore erörtert. Mit dessen Verwendung ist es einem App-Entwickler möglich, sensitives Schlüsselmaterial unter Verwendung hardwaregestützter Schutzmaßnahmen wie dem *Trusted Execution Environment* oder aber einem *Secure Element* abzulegen, sofern diese auf dem Smartphone zur Verfügung stehen und aktiviert sind. Danach sind wir auf die Problematik einer möglichen Rechteausweitung über Apps hinweg eingegangen. Wir haben insbesondere die horizontale Rechteausweitung besprochen, bei der verschiedene Apps Funktionen und Daten anderer Apps erhalten, ohne dass dies ursprünglich so beabsichtigt gewesen wäre. Auch wenn bereits eine ganze Reihe von Ansätzen zur Abschwächung einer horizontalen Rechteausweitung zwischen Apps existieren, so zeigt sich doch ihr begrenzter Nutzen. Insbesondere bei einer zeitlich versetzten IPC-Kommunikation zwischen Apps über einen persistenten Speicher stoßen solche Lösungen doch recht schnell an ihre Grenzen. Wir haben das Kapitel über das Betriebssystem Android abgeschlossen indem wir auf das Schadenspotenzial hingewiesen haben, das mit einer Aktivierung der Kommunikationstechnologie Near Field Communication (NFC) einhergehen kann. Da unter Android die Aktivierung von NFC die Voreinstellung ist, nimmt ein Smartphone in der Nähe eines NFC-Tags dessen ausgesendete NDEF-Nachrichten entgegen und versucht, diese auszuwerten. Daher kann nicht ausgeschlossen werden, dass über diesen Kanal Angriffe auf ein Smartphone vorbereitet werden, falls ein Angreifer ein NFC-Tag so beschreibt, dass eine gesendete NDEF-Nachricht Aktivitäten auf dem Smartphone anstößt, die bösartig sind oder das Sicherheitsniveau heruntersetzen.

Literatur

1. Birrell, A.D., Nelson, B.J.: Implementing remote procedure calls. ACM Transactions on Computer Systems **2**(1), 39–59 (1984)
2. Spreitzenbarth, M.: Mobile Hacking. dpunkt (2017)
3. Sun, S.T., Cuardros, A., Beznosov, K.: Android rooting: Methods detection, and evasion. 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (2015)
4. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. J. Soc. Ind. Appl. Math. **8**, 300–304 (1960)
5. Merkle, R.: Method of providing digital signatures. US4309569A (1979)
6. Tolvanen, S.: Strictly enforced verified boot with error correction. https://android-developers.googleblog.com/2016/07/strictly-enforced-verified-boot-with.html. Zugegriffen: 1. Juli 2019
7. Tanenbaum, A.S., van Steen, M.: Verteilte Systeme – Prinzipien und Paradigmen. Pearson Studium, 2. aktualisierte Auflage (2007)
8. Google Developers: Key and ID attestation, https://source.android.com/security/keystore/attestation. Zugegriffen: 1. Juli 2019
9. Dimitriados, A., Drosatos, G., Efraimidis, P.S.: How much does a zero-permission Android app know about us? In: Central European Cybersecurity Conference (CECC '19), Munich (2019)
10. Biskup, J.: Security in Computing Systems – Challenges, Approaches and Solutions. Springer, Berlin (2009)
11. SEAndroid: http://selinuxproject.org/page/SEAndroid. Zugegriffen: 1. Juli 2019
12. Google Developers: Android keystore system. https://developer.android.com/reference/java/security/KeyStore. Zugegriffen: 1. Juli 2019
13. Sureda, M.O.: Android keystore: What is the difference between "strongbox" and "hardware-backed" keys? https://proandroiddev.com/android-keystore-what-is-the-difference-between-strongbox-and-hardware-backed-keys-4c276ea78fd0. Zugegriffen: 1. Juli 2019
14. Markmann, T., Gessner, D., Westhoff, D.: QuantDroid: Quantitative Approach towards Mitigation Privilege Escalation on Android. In: IEEE ICC, Communication and Information Security Symposium, 2144–2149 (2013)
15. Enck, W., Gilbert, P., Chun, B., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. OSD'10, S. 393–407 (2010)
16. Mollus, K., Westhoff, D., Markmann, T.: Curtailing Privilege Escalation Attacks over Asynchronous Channels on Android. 4CS, 2014: 87–94 (2014)
17. Verdult, R., Kooman, F.: Practical Attacks on NFC Enabled Cell phones. Third International Workshop on Near Field Communication (2011)
18. Whitwam, R.: Verified boot in Android 7.0 won't let your phone boot if the software is corrupt. https://www.androidpolice.com/2016/07/20/verified-boot-android-7-0-wont-let-phone-boot-software-corrupt/.Zugegriffen: 1. Juli 2019

Umsetzung sicherheitskritischer Anwendungen

8

8.1 Mobile Banking-Verfahren mittels App

In den vorangehenden Abschnitten haben wir einige grundsätzliche Dinge über das Rechtemodell von Android erfahren, und uns mit der Problematik der horizontalen Rechteausweitung zwischen Apps vertraut gemacht. Nun wollen wir eine Klasse von Smartphone-Anwendungen ein wenig detaillierter betrachten, für die die Notwendigkeit einer sicheren Umsetzung und passgenauen Ausgestaltung für jedermann offensichtlich ist: mobile Banking-Verfahren auf der Grundlage von Apps. Mobiles Banking, also das Ausführen von Bankgeschäften mithilfe eines Mobiltelefons oder PDAs, erfreut sich einer wachsenden Beliebtheit. Werden die Bankgeschäfte über ein Smartphone abgewickelt, so kann dies wahlweise über einen Browser oder eigens hierfür vorgesehener Apps erfolgen. Neben weiteren technischen Aspekten, auf die wir später noch gesondert eingehen werden, ist bei der Umsetzung von mobilen Banking-Verfahren mittels Apps die sichere Verwendung von Einmalkennwörtern, den sogenannten Transaktionsnummern (TAN), von entscheidender Bedeutung, wobei SMS-TAN und push-TAN die wohl relevantesten TAN-Verfahren sein dürften. Doch zunächst folgen einige Anmerkungen zu den Richtlinien über Zahlungsdienstleistungen, bevor wir uns den hieraus erwachsenden technischen Anforderungen widmen.

8.1.1 Richtlinie über Zahlungsdienstleistungen

Mit der Payment Service Directive 2 (PSD 2) [6] hat die EU eine Richtlinie über Zahlungsdienstleistungen verabschiedet, die im Januar 2018 in Kraft getreten ist und seitdem in regionales Recht umgesetzt werden muss. Neben anderen Aspekten enthält die PSD 2 Vorgaben zur Sicherheit von Zahlungsdienstleistungen, also zur Sicherheit

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_8

von Online-Zahlungen. So wird in Artikel 97 der PSD 2 eine ‚starke Kundenauthentifizierung' gefordert, die dadurch zu gewährleisten ist, dass mindestens zwei Kategorien aus den Bereichen Wissen, Besitz und Inhärenz – also etwas, was der Nutzer ist – unterstützt werden. Diese müssen voneinander unabhängig sein derart, dass die Nichterfüllung eines Kriteriums die Zuverlässigkeit der übrigen Kriterien nicht außer Kraft setzt. Die PSD 2 widmet sich jedoch selbst nicht der Beschreibung der technischen Umsetzung derartiger Anforderungen. Dies erfolgt in der gesonderten Verordnung (EU) 2018/389 [5] als Ergänzung zur PSD 2. Letztere trat im März 2018 in Kraft und sollte bis zum September 2019 von den Mitgliedsstaaten der EU umgesetzt worden sein. Hierin sind die technischen Anforderungen an eine Authentifizierung und an eine gesicherte Kommunikation detaillierter beschrieben. So wird zum einen ein Transaktionsüberwachungsmechanismus gefordert, mit dem nicht autorisierte Zahlungsvorgänge zu erkennen sind. Ebenfalls ist die Sperrung des Accounts bei fünf fehlgeschlagenen Authentifizierungsvorgängen gefordert.

Soll das mobile Banking nutzerseitig allein über ein Smartphone erfolgen, im Jargon der Verordnung (EU) 2018/389 nunmehr Mehrzweckgerät genannt, so sind seitens der Zahlungsdienstleister Maßnahmen zur Risikominderung zu gewährleisten, welche die Unabhängigkeit der Komponenten einer starken Kundenauthentifizierung bestärkt. Im Einzelnen sind dies die Forderungen aus Artikel 9/2:

a) „Nutzung getrennter sicherer Ausführungsumgebung durch die im Mehrzweckgerät installierte Software;
b) Mechanismen, mit denen sichergestellt wird, dass die Software oder das Gerät vom Zahler oder einem Dritten nicht verändert wurde;
c) Sofern Veränderungen stattgefunden haben, Mechanismen zur Eindämmung von deren Folgen."

8.1.2 SMS-TAN und push-TAN

Auch wenn sich verschiedenste TAN-Verfahren für die Durchführung von Transaktionen beim allgemeinen Onlinebanking etabliert haben, so wollen wir hier nur auf die Verfahren SMS-TAN und push TAN eingehen. Beide Verfahren werden gegenwärtig auch für einen Einsatz im mobilen Banking mittels Apps diskutiert und/oder verwendet.

8.1.2.1 SMS-TAN

Beim ursprünglichen SMS-TAN-Verfahren ist das Mobiltelefon als zweiter Faktor gedacht. Der Kunde besucht an seinem PC die Onlinebanking-Webseite der Bank und fordert über die Webanwendung die TAN am PC an. Die Bank stellt dem Kunden nun die TAN auf dessen Mobiltelefon zu, wobei die Übertragung der TAN per SMS erfolgt. Die auf seinem Mobiltelefon erhaltene TAN kann der Kunde nun an seinem PC zur Durchführung der Transaktion verwenden. Diese Abfolge ist in der Abb. 8.1 dargestellt.

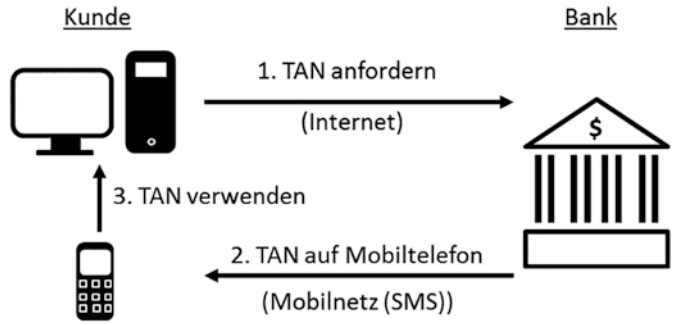

Abb. 8.1 Klassisches SMS-TAN-Verfahren. (Quelle: eigene Darstellung in Anlehnung an [1])

Soweit, so gut. Allerdings stammt das SMS-TAN-Verfahren aus einer Zeit, als das Mobiltelefon tatsächlich vorrangig zum Telefonieren verwendet wurde. Insbesondere gab es nicht die Möglichkeit, einen Webbrowser auf dem Mobiltelefon oder einzelne Apps zu starten. Dies ist jedoch bei einem Smartphone der Fall. Damit führt der Einsatz von SMS-TAN auf einem Smartphone zwangsläufig dazu, dass der zweite Faktor auf demselben Gerät empfangen wird. Damit ergeben sich für den Nutzer einer Banking-App mit TAN-SMS auf einem Smartphone folgende Sicherheitsrisiken: Sollte das Smartphone mit Schadcode infiziert sein, so bestünde ein Zugriff auf die TANs und diese könnten beispielsweise über die Internetverbindung ausgelesen werden. Zudem bestünde die Gefahr, dass, bei Nutzung älterer Mobilfunknetze wie GSM zur Übertragung der SMS, und Verwendung von als gebrochen geltenden Verschlüsselungsverfahren die TANs leicht extrahiert werden können.

8.1.2.2 push TAN

Mit dem pushTAN-Verfahren werden die einzelnen TANs nicht mehr per SMS versendet. Deren Übertragung erfolgt nun über das Internet. Bei den pushTAN-Verfahren können wir des Weiteren zweierlei Ansätze unterscheiden. Zu der ersten Kategorie gehören Verfahren, bei denen der Empfang der TANs und das eigentliche Banking über zwei gesonderte Apps erfolgen. Daneben existieren aber auch pushTAN-Lösungen, die die Handhabung der TANs und das eigentliche Banking über die gleiche App abwickeln, wobei Verfahren dieser Kategorie oftmals die TAN dem Nutzer nicht einmal mehr offenbaren. Beide Push-TAN-Varianten sind in der Abb. 8.2 dargestellt. Damit sind die konzeptionellen Sicherheitsprobleme von pushTAN gegenüber SMS-TAN keineswegs behoben: Für beide Klassen von Verfahren lässt sich festhalten, dass beide Faktoren, die zum Gelingen einer starken Kundenauthentifizierung einbezogen werden, auf demselben Gerät ankommen und dort vorliegen. Im Falle eines korrumpierten Gerätes wären sie daher durch einen Angreifer abgreifbar. Auf einem Android-System würde bei der ersten pushTAN-Variante der Zygote-Prozess zwei ART-Umgebungen instanziieren,

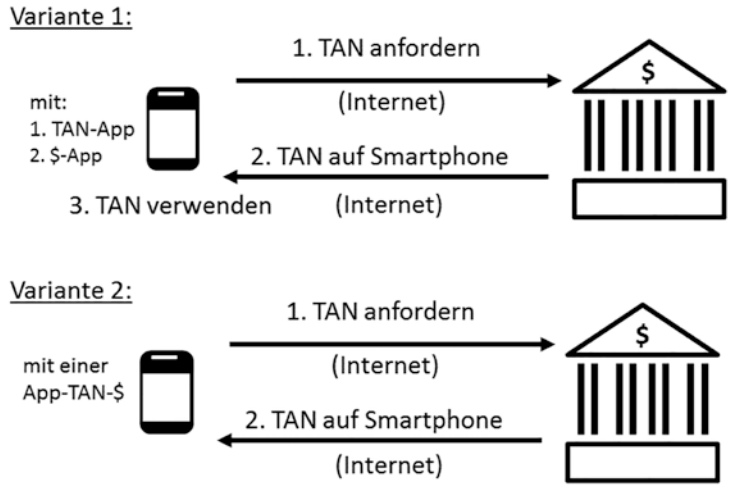

Abb. 8.2 pushTAN-Variante 1 mit separaten Apps für TAN-Empfang (TAN-App) und Banking ($-App) und Variante 2 mit einer einzigen App für TAN-Empfang und Banking (TAN-$-App). (Quelle: eigene Darstellung in Anlehnung an [1])

eine für die TAN-App, eine weitere für die Banking-App, während bei der zweiten pushTAN-Variante der Zygote-Prozess eine ART für die integrierte Banking-App instanziiert. Zygote übergibt die Anwendungsdaten an den neuen Prozess. Dabei teilt sich der neu ‚geforkte' Prozess den dynamischen Speicher (Heap) solange mit Zygote, bis eine hierin gestartete App, im Falle des mobile Banking-Verfahrens also entweder die TAN-App, die $-App oder die TAN-$-App, ihre Daten erstmalig auf den Heap schreibt. Erst dann erhält die ART mitsamt der jeweiligen App ihren eigenen zusammenhängenden Speicherabschnitt für ihren dynamischen Speicher (siehe Abschn. 7.1). Werden eine TAN-App und eine $-App verwendet, so ist eine IPC-Variante zwischen Komponenten dieser Apps notwendig, um die Informationen auszutauschen, wie dies bereits in Abschn. 7.2.2 beschrieben wurde. Für den Fall, dass eine einzige TAN-$-App Verwendung findet, ist dies nicht erforderlich.

Will ein Angreifer nun eine TAN-App mitsamt ihren Daten auf ein anderes Gerät kopieren, so gelingt ihm dies, sofern er Root-Rechte auf dem System erlangt hat. Die nachfolgend erörterten Ansätze zur Erkennung von Root-Rechten bzw. die Einbeziehung des Fingerprints des Gerätes zielen darauf ab, ebendies zu verhindern.

8.1.3 Sicherheitsvorgaben für TAN-Apps

Aus der PSD 2 und der Verordnung (EU) 2018/389 ergeben sich nun die folgenden konkreten technischen Sicherheitsvorgaben für eine zu implementierende TAN-App:

1. Technische Umsetzung einer Root-Erkennung
2. Technische Umsetzung von Zertifikats-Pinning
3. Sichere Ausführungsumgebung

Im Rahmen einer Ausarbeitung im Modul Security Trends im Studiengang Unternehmens- und IT-Sicherheit (UNITS) der Hochschule Offenburg hat der Bachelorstudent Max Bauert im Januar 2019 bestehende TAN-App-Lösungen hinsichtlich der Umsetzung der obigen Kriterien untersucht. Auf die Ergebnisse seiner Ausarbeitung beziehen sich die nachfolgenden Abschnitte zu Root-Erkennung, einem Zertifikats-Pinning sowie Betrachtungen zu einer sicheren Ausführungsumgebung. Mit einer TAN-App sollen TANs sicher auf einem Smartphone oder Tablet, also einem Mehrzweckgerät, empfangen werden können. Nur wenn alle drei der technischen Sicherheitsvorgaben so umgesetzt worden sind, dass sie nicht umgangen werden können, kann man von einer PSD-2- sowie (EU)-2018/389-konformen, sicheren TAN-App-Lösung sprechen.

8.1.3.1 Root-Erkennung

Die von den Kreditinstituten angebotenen TAN-Apps müssen über eine Root-Erkennung verfügen. Die Root-Erkennung überprüft, ob der Nutzer auf dem Gerät administrative Rechte (Root-Rechte) besitzt. Man muss sich klarmachen: Der Nutzerprozess der TAN-App, der unter vollständiger Kontrolle des Betriebssystems und der Laufzeitumgebung steht, ist laut Sicherheitsvorgaben angehalten, in Erfahrung bringen zu können, ob der Nutzer administrative Rechte besitzt. Allerdings gilt es Folgendes zu bedenken: Hat der Nutzer diese, so kann er jedem sich in der Ausführung befindenden Programm – also auch der TAN-App – vorgaukeln, eben nicht über diese zu verfügen. Oder anders formuliert: Die erste konkrete technische Sicherheitsvorgabe, die Erkennung von Root-Rechten auf einem Smartphone oder Tablet durch einen Nutzerprozess, ist prinzipiell nicht umsetzbar. Damit steckt die mobile Banking Branche, die TAN-Apps auf Smartphones propagiert, in einem Dilemma: Diese für die Sicherheit von TAN Verfahren *notwendige,* wenn auch noch nicht hinreichende, Vorgabe zur Gewährleistung eines sicheren mobilen Bankings ist prinzipiell nicht erfüllbar!

Und dennoch versuchen die Entwickler diese Vorgabe zu erfüllen, beispielsweise indem sie zumindest die API *SafetyNet Attestation* des Unternehmens Google hierfür verwenden. Über diese Schnittstelle können Informationen zur vorhandenen Software und Hardware des Gerätes gesammelt und zu einem Profil zusammengestellt werden.

Seriöse Entwickler von TAN-Apps werden in aller Regel jedoch noch weitere Überprüfungen zur Root-Erkennung implementieren. Allerdings sind auch diese, wie wir sehen werden, allesamt mehr oder weniger leicht zu umgehen [7]:

1. Ist ein `su`-Binary vorhanden? Das Kommando `su` steht für substitute user identity. Ohne Angabe eines Benutzernamens wird nach Passworteingabe zum Benutzer root gewechselt.
2. Ist eine App zur Verwaltung von Root-Rechten auf dem System vorhanden?

3. Ist ein BusyBox-Binary mit Standard-Unix-Dienstprogrammen vorhanden oder andere APK-Dateien?
4. Überprüfe BUILD-Tag.
5. Sind Berechtigungen in Verzeichnissen geändert worden?
6. Überprüfe die Liste der aktuell laufenden Prozesse mit Root-Rechten.
7. Führe einzelne Shell-Befehle aus und interpretiere deren Rückgabe. Hierzu bieten sich insbesondere die Kommandos `su` und `which su` an.

Allerdings hat sich für Android auch schon eine sehr agile Community gebildet, die darauf abzielt, obige Root-Erkennungsmechanismen auszuhebeln. Aktuell ist Magisk das wohl ausgereifteste Werkzeug, das zur Verwaltung von Root-Rechten genutzt werden kann und einige der obigen Punkte zur Root-Erkennung ungeeignet erscheinen lässt. So kann mit Magisk die Root-Erkennung der SafetyNet Attestation API in jedem Falle umgangen werden. Die Tab. 8.1 fasst die wesentlichen Ergebnisse der Sicherheitsanalysen von Max Bauert [1] zusammen. Hierbei wurde ein Android-Smartphone mit Android-Version 7.1.2 verwendet. Ein Eintrag in einer Zelle der Tabelle bedeutet, dass die TAN-App die Root-Rechte des Nutzers auf der Zielplattform erkennt und daraufhin seinen Start verweigert.

Die Zeile ‚kein Root' bedeutet dabei, dass die TAN-Apps aller Kreditinstitute ausgeführt werden, wenn auf dem Zielgerät keine Root-Rechte des Nutzers vorliegen. Sind jedoch für den Nutzer auf dem Smartphone Root-Rechte vergeben, so müsste die TAN-App seine Ausführung umgehend beenden. Für den Fall, dass die Vergabe der Root-Rechte mit `su`-Binary erfolgte, erkennen fast alle TAN-Apps dies und beenden sich automatisch. Allerdings handelten auch hier zwei Apps nicht PSD -2- bzw. (EU)-2018/389-konform. Kommt hingegen Magisk zum Einsatz, so funktioniert die Root-Erkennung nur noch bei zwei Anbietern von mobilen Banking-Verfahren mittels TAN-App. So bleibt nur zu hoffen, dass dieses desaströse Zwischenfazit bis zum September 2019 vorteilhaft korrigiert wurde. Allerdings besteht hierzu aufgrund meiner eingangs genannten Anmerkungen wenig Hoffnung. Denn auch wenn seitens der Entwickler von TAN-Apps immer findigere Indizien zur Root-Erkennung umgesetzt werden, so ist

Tab. 8.1 Test der Root-Erkennung verschiedener TAN-Apps (Quelle: eigene Darstellung in Anlehnung an [1])

	Sparkasse pushTAN	DKB tan2go	BW-Bank BW-push-TAN	Deutsche Bank DWS Secure TAN	pbb direkt pbb direkt pushTAN	comdirekt comdirekt	Volksbank SecureGo
kein Root							
su Binary	○	○	○		○		○
Magisk	○						○

es nur eine Frage der Zeit, bis diese in Werkzeugen wie Magisk oder anderen zur Aushebelung der Root-Erkennung berücksichtigt werden.

▶ Konkrete technische Sicherheitsvorgaben der EU-Verordnung 2018/389 wie die der Root-Erkennung oder eines Geräte-Fingerprinting sind eine zwingende Voraussetzung, um für das mobile Banking TAN-Apps auf Smartphones oder Tablets sicher ausführbar zu machen und diese nicht so zu manipulieren, dass sie auf einem anderen Gerät zu starten wären. Da insbesondere die technische Umsetzung einer Root-Erkennung allein auf der Implementierung von ‚Indizien' über vergebene Root-Rechte eines Nutzers basiert, die von einer TAN-App als ein Nutzerprozess zu erfassen sind, können diese dem Nutzerprozess jederzeit durch Modifizierungen der Ausführungsumgebung bzw. unter Verwendung der Open-Source-App Magisk der TAN-App vorgegaukelt werden.

8.1.3.2 Gerätebindung und sichere Anzeige der Daten

In [4] weisen die die Autoren darauf hin, dass App-basierte mobile Banking-Verfahren in Bezug auf einen praktischen Kopierschutz von unmöglich über mäßig bis vollständig reichen. So basieren die gegenwärtigen verfügbaren Verfahren zur Gerätebindung auf Algorithmen zum Geräte-Fingerprinting, bei denen idealerweise viele Werte verwendet werden welche einerseits *stabil* und andererseits *einzigartig* sind. Allerdings zeigt sich eine Tendenz dahingehend, dass die Hersteller oftmals eher defensivere Varianten umsetzen und daher nur wenige charakteristische Gerätewerte für das Geräte-Fingerprinting verwenden. Dies führt teilweise dazu, dass einzelne Verfahren stabil laufen, jedoch das eigentliche Ziel, die Prüfung der Einzigartigkeit, nicht in vollem Umfang gewährleistet ist.

Seit dem Erscheinen von Android 6 und iOS9 sind nunmehr Gerätegenerationen auf dem Markt, die einerseits durch einen hardwaregestützten Keystore (siehe Abschn. 7.4.2) oder andererseits mittels KeyChain und SecureEnclave über spezielle Hardwarebausteine, ein sogenanntes *trusted execution environment* (TEE), verfügen. Somit wäre durch eine gesicherte Ablage eines privaten Schlüssels der Einsatz von asymmetrischer Kryptografie zum Geräte-Fingerprinting ohne Weiteres auf diesen Geräteklassen möglich, mit dem dann ein zweifelsfreier praktikabler Kopierschutz erreichbar wäre. Allerdings haben die technischen Regulierungsstandards den Einsatz einer separaten vertrauenswürdigen Ausführungsumgebung (TEE) für das Device-Fingerprinting nicht verbindlich vorgegeben, sodass es den Herstellern freigestellt ist, auf diese Möglichkeit zurückzugreifen.

In [4] weisen die Sicherheitsexperten noch auf einen weiteren kritischen Punkt einer Sicherheitsarchitektur für mobile Banking-Apps hin, dessen Umsetzung weitgehend ungelöst ist. Die Verifizierbarkeit der Transaktionsdetails muss dem Nutzer einer mobilen Banking-App garantiert sein. Im Klartext: Die Daten die der Nutzer einer Banking-App sieht – die also von der Banking-App dargestellt werden –, sind

tatsächlich diejenigen die schlussendlich dem Kreditinstitut nach Abwicklung der geldwerten Transaktion vorliegen. Die dargestellten Daten müssen also mit den empfangenen übereinstimmen und dürfen danach auch nicht mehr unerkannt veränderbar sein. Die Gewährleistung solch einer sicheren Anzeige der Transaktionsdaten ist aber mitnichten gegeben.

8.1.3.3 Zertifikats-Pinning

Die Kommunikation zwischen der TAN-App und der serverseitigen Banking-Anwendung wird über das Protokoll *Hypertext Transfer Protocol Secure* (HTTPS) abgesichert. Dieses Protokoll hat sich bei der Absicherung der Kommunikationsverbindungen von Webanwendungen etabliert und bietet einen Ende-zu-Ende-Schutz zwischen den clientseitig und serverseitig laufenden Prozessen hinsichtlich der Schutzziele *Vertraulichkeit, Authentizität* und *Integrität.* Erreicht wird dies durch die Verwendung des Protokolls Transport Layer Security (TLS), dessen Sicherheit maßgeblich auf der Verwendung von Zertifikaten beruht.

Dabei dient die eigentliche Funktion eines digitalen Zertifikates dem Binden eines öffentlichen Schlüssels $k_{pub,A}$ einer Partei Alice (A) an ihre Identität ID_A, also $(k_{pub,A}, ID_A)$. Niemand könnte unseren Angreifer Mallory (M) daran hindern, dieses 2-Tupel zu manipulieren, beispielsweise in $(k_{pub,M}, ID_A)$. Er fügt also seinen eigenen öffentlichen Schlüssel anstelle des öffentlichen Schlüssels von Alice ein, lässt die Identität von Alice jedoch unangetastet. Da ein öffentlicher Schlüssel im Wesentlichen eine Binärfolge ohne erkennbare Struktur ist, wäre der Angreifer Mallory von nun an in der Lage jegliche Kommunikation an Alice über HTTPS zu belauschen und mit seinem privaten Schlüssel zu entschlüsseln. Eigentlich sind die Dinge noch ein wenig komplizierter, aber für den Moment soll diese Betrachtungsweise erst einmal ausreichend sein.

Diese Vorüberlegungen lassen uns die Sinnhaftigkeit eines Zertifikatsaufbaus besser nachvollziehen, wobei wir in der nachfolgenden schematischen Darstellung bewusst auf die Erörterung weiterer notwendiger Details wie den Gültigkeitszeitraum des Zertifikats, die zum Einsatz kommenden Algorithmen, sowie die genauen Parameter des Schlüssels verzichten. X.509- Zertifikate sind Standard für SSL- oder TLS-Verbindungen. Uns reicht die folgende Notation, wobei $sig_{pr,CA}()$ die digitale Signatur mit dem privaten Schlüssel $k_{pr,CA}$ einer Zertifikatsstelle oder Certificate Authority (CA) ist:

$$Zert_A = [(k_{pub,A}, ID_A),\ sig_{pr,CA}(k_{pub,A}, ID_A)$$

Da der Ausdruck $(k_{pub,A}, ID_A)$ wie oben gesehen manipulierbar ist, geht in das Zertifikat also eine digitale Signatur ein, wobei die Signatur gebildet wird über den Daten $k_{pub,A}$ und ID_A, und dem privaten Schlüssel $k_{pr,CA}$ einer beiderseitig vertrauenswürdigen Partei, der Zertifikatsstelle.

Der alleinige Zweck von Zertifikaten ist also das Binden der Identität eines Nutzers an dessen öffentlichen Schlüssel.

Nun sind die Dinge auch hier wie so oft ein wenig komplizierter. Denn möchte eine Partei nun den öffentlichen Schlüssel $k_{pub,A}$ von Alice tatsächlich verwenden, so muss

sie die Gültigkeit des Zertifikates prüfen, indem sie eine Signaturverifikation durchführt. Hierzu wir allerdings der öffentliche Schlüssel der CA benötigt und, man ahnt es schon, auch dieser muss mit dem Namen der CA gebunden in Form eines weiteren Zertifikates vorliegen. Und so weiter und so fort. Dies ist der Grund, weshalb es Publice-Key-Infrastrukturen (PKI) oder andere Strukturen zur Verifizierung von Zertifikaten gibt.

Nach diesen einleitenden Bemerkungen über Zertifikate nun aber zurück zum Wesen von Zertifikats-Pinning, das ursprünglich innerhalb der IETF [2] für Browser-basierte Webapplikationen propagiert wurde und nun durch die PSD 2 und die Verordnung (EU) 2018/389 als konkrete technische Sicherheitsvorgabe für eine zu implementierende TAN-App gefordert ist.

Zertifikats-Pinning ist ein Ansatz zur Absicherung gegen MitM-Angriffe für den Fall, dass der Angreifer über ein Zertifikat einer CA verfügt, der das Opfer vertraut. Im Falle von Zertifikats-Pinning wird nicht mehr der CA vertraut, sondern es wird nur noch bekannten öffentlichen Schlüsseln oder Zertifikaten vertraut. Das Verfahren basiert auf der Annahme *Trust-on-First-Use* (ToFu) und bindet den öffentlichen Schlüssel oder das Zertifikat der Zwischen-CA oder Root-CA an einen *HTTP Public Key Pinning-(HKPK-)*Pin, der auf der TAN-App nach dem ersten Kontakt zum Bank-Server gespeichert wird. Mit jedem weiteren Kontakt zum Bank-Server geht dieser HKPK-Pin nun wieder mit in die Anfrage ein. Hierdurch soll verhindert werden, dass ein Angreifer sein eigenes, von einer CA zertifiziertes, falsches Zertifikat für den Angriff verwenden kann. Der prinzipielle Ablauf von Public-Key-Pinning ist in Abb. 8.3 dargestellt. Dabei hat ein *public key pin* (*PKP*) laut dem Dokument *Request for Comments* (RFC) mit der Nummer 7469 der IETF folgende Bauart:

Public-Key-Pins:
max-age=”…”;
pin-sha256=“…”;
pin-sha256=“…”;
report-uri=“http://example.com/pkp-report”

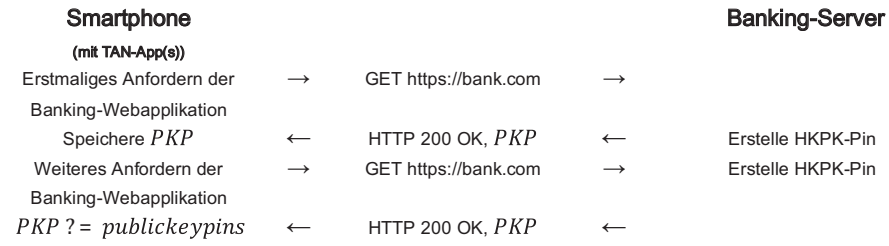

Abb. 8.3 Zertifikats-Pinning für mobile Banking-Verfahren unter *Trust-on-First-Use*-Annahme. (Quelle: eigene Darstellung)

Aktuell sieht der RFC als alleinig gültige Hashfunktion SHA-265 vor. Der Wert max-age legt die maximale Gültigkeitsdauer fest. Optional kann darüber hinaus über report-uri festgelegt werden, an welche Adresse im Falle einer fehgeschlagenen PKP-Validierung dieses Ereignis zu berichten ist.

Angriffsmöglichkeiten sind aber auch bei Verwendung von Public-Key-Pinning nicht vollständig auszuschließen: Insbesondere geht die ToFU-Annahme davon aus, dass bei erstmaligem Kontakt kein MitM-Angriff erfolgen darf. Dieser wäre auch durch ein Zertifikats-Pinning nicht zu unterbinden. Denn schafft es ein Angreifer demnach den Übertragungsverkehr zwischen Banking-App oder Browser-basierter clientseitiger Anwendung und Bank-Server zu stören, und bringt er das Opfer dazu, den Kontakt zum Bank-Server von einem anderen Gerät bzw. Browser zu initiieren, so wird dies als erstmaliger Kontakt interpretiert, der HKPK-Pin ist per MitM abgreifbar und die verschlüsselte Verbindung kann dauerhaft aufgebrochen werden.

Neben all dem Gesagten verfügt Android über verschiedene Zertifikatsspeicher, und zwar einen Speicher für die Benutzerzertifikate, und einen weiteren für die Systemzertifikate. So kann ein Benutzer eigene CA-Zertifikate ausschließlich im Speicher für Benutzerzertifikate ablegen. Allerdings wurden in früheren Android-Versionen Zertifikaten aus beiden Speichern vertraut. Mit Android 7 werden von Apps allerdings nur noch Systemzertifikaten vertraut (siehe Abschn. 7.3.2). Dies soll zu einer deutlichen Erhöhung der Sicherheit gegenüber MitM-Angriffen führen. Eigentlich kann ein Benutzer damit seine Zertifikate nicht mehr ohne Weiteres in den Speicher für Systemzertifikate ablegen. Mit den richtigen Werkzeugen ist dies aber dennoch möglich wie wir gleich erfahren werden.

8.1.3.4 Banking-Apps mit einer UID

Wie wir bereits aus dem Abschn. 7.4 wissen, erhält unter Android eine App bei ihrer Installation eine feste UID, die entwicklerspezifisch ist. Die App bekommt hierüber entsprechend dem Android -Zugriffsmodell verschiedenste Ausführungsrechte auf eigene Dateien und andere Ressourcen. Jede App, die mit dem gleichen Entwicklerschlüssel signiert ist, erhält auf dem Smartphone auch die gleiche UID und die gleichen Ausführungsrechte auf ebendiese Ressourcen. Wäre es einem Angreifer nun möglich, eine App zu entwickeln die das Entwicklerzertifikat der Banking-App nutzt, so hätte er hierüber unter Umständen Zugriff auf deren sensible Banking-Daten. Auch wenn dies eher ein Angriff ist, der Insider-Wissen erfordert, so soll er an dieser Stelle doch nicht unerwähnt bleiben. In diesem Zusammenhang ist es möglicherweise auch nicht uninteressant, zu wissen, dass Banking-Apps zum Teil von externen Firmen implementiert werden, die typischerweise auch die Banking-Apps weiterer Kreditinstitute umsetzen.

8.1.3.5 Werkzeuge zur Umgehung von Zertifikats-Pinning

Als Werkzeuge, mit denen das Zertifikats-Pinning umgangen werden kann, eignen sich Magisk und Frida [3]. So kann man beispielsweise mit der Hilfe von Magisk Benutzerzertifikate in den Speicher für Systemzertifikate verschieben, damit solchen Zertifikaten

dann fälschlicherweise wieder vertraut wird. Frida sollte zusammen mit der Erweiterung Objection eingesetzt werden. Frida dient dem Zuschalten (engl. *hook*) und Manipulieren von Funktionsaufrufen mit Hilfe von JavaScript. Hierdurch lässt sich fremder Code in den bestehenden Code einfügen, beispielsweise um deren Ablauf zu verändern oder um Ergebnisse abzufangen. Eigene Skripte können ebenfalls verwendet werden. Mit Frida kann über eine Shell ein Server auf einem Android-Gerät oder einem iOS-Gerät gestartet werden, der mit einem Client auf einem PC kommuniziert. Objection liefert dazu das passende Runtime Exploration Framework.

Max Bauert [1], ein Student der Hochschule Offenburg, hat die obigen Werkzeuge in einer Laborumgebung verwendet, um einen MitM-Angriff auf die Banking-Apps der DKB und von comdirect durchzuführen. In beiden Fällen hat das in den TAN-Apps umgesetzte Zertifikats-Pinning dem Angriff jedoch standgehalten. Da die Apps darüber hinaus einer Code-Obfuskierung unterzogen wurden, steht zu vermuten, dass das Brechen des Zertifikats-Pinning mit Verfahren des Reverse Engineering eine weitere Hürde setzt. Letztere Untersuchung wurde jedoch nicht weiter durchgeführt. Die verbreitete Nutzung von Zertifikats-Pinning bei der Erstellung von Apps, bei denen eine gesicherte Kommunikation zu einem Server aufgebaut wird, ist Anlass genug, dass wir uns diesem Thema in Kapitel 9 noch einmal gesondert widmen werden.

8.2 Zusammenfassung

In diesem Kapitel haben wir uns mit den Anforderungen einer Umsetzung sicherheitskritischer Anwendungen auf einem Smartphone beschäftigt. Hierzu haben wir mobile Banking-Verfahren mittels Apps als Referenzanwendung betrachtet. Mit der Richtlinie über Zahlungsdienstleitungen, der Payment Service Directive 2 (PSD 2) und der gesonderten Verordnung (EU) 2018/389 als Ergänzung zur PSD 2 sind technische Sicherheitsvorgaben für eine zu implementierende TAN-App vorgegeben. Neben einer Root-Erkennung und dem Zertifikats-Pinning wird hier auch eine sichere Ausführungsumgebung bei der Umsetzung einer TAN-App gefordert. Allerdings ist es weitgehend den Entwicklern überlassen, wie genau diese Forderungen umzusetzen sind. Wir haben erörtert, dass ein Entwickler prinzipiell eine ganze Reihe von Maßnahmen zur Root-Erkennung programmieren kann, wobei diese allesamt mehr oder weniger leicht zu umgehen sind. Da insbesondere die technische Umsetzung einer Root-Erkennung allein auf der Implementierung von Indizien über vergebene Root-Rechte eines Nutzers basiert, die von einer TAN-App als ein Nutzerprozess zu erfassen sind, können diese dem Nutzerprozess aber jederzeit durch Modifizierungen der Ausführungsumgebung bzw. unter Verwendung der Open-Source-App Magisk vorgegaukelt werden. Während für das Geräte-Fingerprinting prinzipiell Lösungen durch strikte Verwendung kryptografischer Lösungen im Zusammenspiel mit dem Android Keystore und einer Anbindung an eine vertrauenswürdige Hardware denkbar sind, ist die Problematik der technischen Umsetzung einer sicheren Anzeige der Transaktionsdaten auf dem Smartphone noch

gänzlich ungeklärt. Denn die dargestellten Daten müssen mit den empfangenen übereinstimmen und dürfen nach der Betrachtung durch den Nutzer auch nicht mehr unerkannt veränderbar sein. Wir haben auch auf die besondere, hoffentlich nur theoretische Problematik hingewiesen, falls verschiedene Banking-Apps mit dem gleichen Entwicklerzertifikat ausgestellt worden sein sollten. Diese hätten die gleiche UID und entsprechend dem Rechtemodell auch Zugriff auf die gleichen Ressourcen. Da neben der Verwendung bei mobilen Banking-Verfahren die Technik des Zertifikats-Pinning noch bei einer ganzen Reihe weiterer mobiler Anwendungen mit gesicherter Anbindung an einen Server zum Einsatz kommt, wollen wir einige etablierte Techniken zum Zertifikats-Pinning im folgenden Kapitel ein wenig genauer erörtern. Insbesondere interessiert uns hierbei, welche Möglichkeiten Mallory besitzt, eine derart geschützte Ende-zu-Ende -Verbindung dennoch aufzubrechen.

Literatur

1. Bauert, M.: Analyse App-basierter mobile Bankingverfahren für Android und iOS. UNITS, International Security Trends, Hochschule Offenburg (2019)
2. Evans, C., Palmer, C., Sleevi, R.: Public Key Pinning Extension for http. IETF RFC 7469 (2015)
3. Frida: https://www.frida.re. Zugegriffen: 30. Okt. 2019
4. Haupert, V., Pugliese, G.: Die Realität von Mobilebanking zwischen allgemeinen und rechtlichen Anforderungen. Sicherheit 2018, Konstanz, GI-Edition, 171–182 (2018)
5. Richtlinie (EU) 2015/2366 des Europäischen Parlaments und des Rates über Zahlungsdienste im Binnenmarkt (November 2015)
6. Träder, D., Zeier, A., Heinemann, A.: Sichere abgeleitete Identität mithilfe des PSD2. Sicherheit 2018, Konstanz, GI-Edition, 182–194 (2018)
7. Sun, S.T., Cuardros, A., Beznosov, K.: Android rooting: methods detection, and evasion. In: 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (2015)

Techniken des Zertifikats-Pinning

9

9.1 Techniken des Zertifikats-Pinning und deren Umgehung

Im Rahmen seiner Masterarbeit an der Hochschule Offenburg im Studiengang *Enterprise and IT-Security* (ENITS) ist Ulrich Winterer [5] der Frage nachgegangen, welche gängigen Varianten einer Implementierung von Zertifikats-Pinning existieren und welche Techniken es zum Aufbrechen dieser gibt bzw. inwieweit weitere Maßnahmen gegen das Aufbrechen von Zertifikats-Pinning umsetzbar sind. In seinen Untersuchungen geht er von dem mobilen Betriebssystem Android 7 oder aktuelleren Versionen aus. Einige der wesentlichen Ergebnisse seiner Ausarbeitung sollen nun vorgestellt werden.

9.1.1 Varianten der Implementierung von Zertifikats-Pinning

Um für eine Android-Applikation die Technik des Zertifikats-Pinning umzusetzen, sind gleich eine Reihe von Vorgehensweisen denkbar.

Möglichkeit 1 Einbindung innerhalb der network_security_config.xml

Hierzu wird ein Hashwert x_1 des zu verwendenden Zertifikates *Cert* in der Konfigurationsdatei hinterlegt. Zur Anwendung kommt die Hashfunktion $h()$ wie beispielsweise SHA-256. Der Hashwert $x_1 = h(Cert)$ wird innerhalb der Tag-Umgebung <domain-config>… </domain-config> abgelegt. Hierbei ist es vorteilhaft, neben dem Hashwert x_1 für das eigentliche Zertifikat noch einen weiteren Hashwert x_2 für ein sogenanntes Backup-Zertifikat $Cert_{backup}$ einzutragen, also $x_2 = h\left(Cert_{backup}\right)$. Dieser zweite Hashwert kommt immer genau dann zum Einsatz, wenn beispielsweise die Gültigkeitsdauer für das eigentlich zu verwendende Zertifikat überschritten wurde.

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_9

Möglichkeit 2 Verwendung von Bibliotheken

Es existieren eine Reihe von Bibliotheken, die Java-Klassen enthalten, in denen das Zertifikats-Pinning bereits umgesetzt ist. Die vermutlich am häufigsten verwendete Bibliothek dieser Kategorie ist die Bibliothek okhttp [2], in der die Klasse CertificatePinner enthalten ist. Die Methoden dieser Klasse kann nun ein Android-App-Entwickler nutzen, um das Zertifikats-Pinning für seine Applikation relativ komfortabel umzusetzen. Für unsere nachfolgenden Betrachtungen aus der Perspektive von Mallory ist es allerdings gar nicht so entscheidend, wie diese Methoden der Klasse CertificatePinning konkret eingesetzt werden. Es ist vielmehr wichtig, dass diese Bibliotheken überhaupt bei der Implementierung einer App verwendet worden sind.

Möglichkeit 3 Eigenimplementierung

Natürlich bestünde auch immer die Möglichkeit einer Eigenimplementierung des Zertifikats-Pinning durch den jeweiligen Entwickler einer App. Generell ist es das Credo der Sicherheits-Community von Eigenimplementierungen kryptografischer Bausteine oder Sicherheitskomponenten strikt abzuraten, da sich hier bei einer Implementierung sehr schnell Fehler einschleichen können, die dann von findigen Angreifern sicherlich sehr zeitnah für potenzielle Angriffe genutzt würden.

9.1.2 Zertifikatsarten

Nachdem der Programmierer die Entscheidung getroffen hat, in welcher Variante er das Zertifikats-Pinning umsetzen möchte, ist es ebenfalls von einer gewissen Bedeutung, welche Art von Zertifikaten angefügt (engl. *pinning*) werden soll. Hierbei unterscheidet man zwischen Blattzertifikaten, Zwischenzertifikaten sowie Wurzelzertifikaten. Derartige Überlegungen, welche Arten von Zertifikaten angehängt werden sollen, sind stark getrieben von der Beobachtung, dass in dem Moment, in dem die Gültigkeit eines konkreten Zertifikats abgelaufen ist, die gesicherte Kommunikation zwischen der Smartphone-App zum jeweiligen Server nicht mehr durchgeführt werden kann. Was bleibt sind entweder Backup-Zertifikate oder aber eine weichere Eingrenzung. Diese kann erfolgen, indem man nicht nur einzelnen konkreten Zertifikaten vertraut, sondern all jenen, die von einer bestimmten Zertifizierungsstelle (CA) ausgestellt worden sind. Eines sollte jedoch auch offensichtlich sein: Je mehr das Vertrauen auf der Basis derartiger Zertifikatsketten delegiert wird, desto eher können sich auch Zertifikate einschleichen, die von Zertifizierungsstellen ausgestellt wurden, deren Vertrauensvorschub durch nichts zu rechtfertigen ist. Wie bei allen (offenen) PKI-Ansätzen besteht also auch beim Zertifikats-Pinning das nicht gerade geringe Risiko, durch eine bösartige Zertifizierungsstelle unterlaufen worden zu sein, die mit Kenntnis des privaten Schlüssels die gesicherte Verbindung zu jeder Zeit auf der Übertragungsstrecke aufbrechen kann.

Blattzertifikat Wird ein Blattzertifikat beispielsweise durch Eintrag von $x = h\left(Cert_{leaf}\right)$ in die *network_security_config.xml* gepinnt, so bedeutet dies, dass die App nur mit diesem einen Zertifikat eine gesicherte Verbindung zum Server, von dem das Zertifikat stammt, etablieren kann. Ist das Blattzertifikat nach einer gewissen Zeit aufgrund seines Gültigkeitszeitraumes abgelaufen, so lässt sich unwiderruflich auch keine gesicherte Verbindung zum Server mehr herstellen.

Zwischenzertifikat Wird ein Zwischenzertifikat durch Eintrag von $x = h(Cert_{intermediate})$ gepinnt, so können nur gesicherte Verbindungen aufgebaut werden zu solchen Servern, die Zertifikate besitzen, die von dieser Zwischen-CA ausgestellt wurden. Damit wird allen Zertifikaten $Cert_{ZwischenCA}(ID_A, k_{pub}, s)$ vertraut mit $s = Sig_{kpZwischenCA}(h(ID_A, k_{pub}))$.

Wurzelzertifikat Wird ein Wurzelzertifikat durch Eintrag von $x = h(Cert_{root})$ gepinnt, so wird allen Zertifikaten vertraut, die von dieser Wurzel-CA ausgestellt worden sind, also allen Zertifikaten $Cert_{WurzelCA}(ID_A, k_{pub}, s)$, mit $s = Sig_{kpWurzelCA}(h(ID_A, k_{pub}))$. Doch damit nicht genug: Es wird auch denjenigen Zertifikaten vertraut, die von einer Zwischen-CA ausgestellt worden sind, selbst aber von der Wurzel-CA signiert wurden. Ein derartiges Pinning durch Anwendung eines Wurzelzertifikates lässt damit eine gesicherte Verbindung auf Basis einer Vielzahl von Zertifikaten zu.

Durch die hier geschilderten Möglichkeiten einer Ausgestaltung durch den App-Entwickler und der Wahl der Zertifikatsart wird der oftmals zu beobachtende Zielkonflikt der beiden teilweise konkurrierenden Ziele Sicherheit und Verfügbarkeit an diesem konkreten Beispiel sehr deutlich.

9.1.3 Aufbrechen von Zertifikats-Pinning ohne Obfuskierung

Für nicht zusätzlich obfuskierte Apps sind bereits eine ganze Reihe von Skripten entwickelt worden, mit denen sich eine mit Zertifikats-Pinning gesicherte Verbindung aufbrechen lässt. Derartige Skripte nutzen zum Teil unterschiedliche Konzepte. Am bewährtesten sind die nachfolgend aufgeführten:

1. *Die Erstellung eines eigenen Trust-Managers*
   Hierbei ist es das Ziel eines Angreifers, über ein Skript einen eigenen Trust-Manager zu erstellen mit einem eigenen Zertifikat. Wird dieses dann später vom Opfersystem verwendet, so kann der Angreifer, der ja im Besitz des passenden privaten Schlüssels zu dem im Zertifikat enthaltenen öffentlichen Schlüssel ist, jegliche Kommunikation über diese Verbindung entschlüsseln und mitlesen. Dieser Ansatz aus dem Jahre 2017 wurde erstmalig von dem Sicherheitsexperten Piergiovanni Cipolloni [1] öffentlich gemacht.

2. *Das Setzen der bestehenden Trust-Manager-Implementierung auf null*
   Dies ist gleichbedeutend mit dem Löschen der Datei, in der die Netzwerk-Sicherheits-Konfiguration beschrieben ist. Konkret wird die Datei *network_security_config.xml* gelöscht. Dieses Umgehen des Zertifikats-Pinning bei Android wurde im Jahr 2018 von Mattia Vinci [4] vorgestellt.
3. *Überladen von Funktionalität*
   Wieder andere Skripte zielen darauf ab, die Funktionalität derjenigen Klasse zu überladen, die das Zertifikats-Pinning leistet: Liegt die App nicht obfuskiert vor, wird beispielsweise nach der Klasse CertificatePinner der Bibliothek okhttp gesucht. Dann ermöglicht das ClassLoader-Konzept von Java zur Laufzeit, die Funktionalität der Klasse mit einer anderen Funktionalität zu überladen. In diesem konkreten Fall wird die Klasse CertificatePinner mit einer anderen Funktion überladen, um auf diese Weise den Sicherheitsmechanismus auszuhebeln. Auch dieser Vorschlag ist noch relativ jung. Er stammt aus dem Jahre 2018 [3].

9.1.4 Aufbrechen von Zertifikats-Pinning mit Obfuskierung

Im Rahmen seiner Masterarbeit im Studiengang ENITS an der Hochschule Offenburg hat der Student Ulrich Winterer ein Python-Skript zur Umgehung des Zertifikats-Pinning für obfuskierte Applikationen entwickelt welche die oben bereits erwähnte okhttp-Bibliothek nutzen. Denn mehr und mehr Apps die die Technik des Zertifikats-Pinning einsetzen, wurden darüber hinaus vorab obfuskiert. Welches genau die gängigen Verfahren zur Obfuskierung und Deobfuskierung von Apps sind, wird uns in Kap. 10 noch eingehender beschäftigen. Zum jetzigen Zeitpunkt reicht es zum Verständnis völlig aus, dass mittels Obfuskierungstechniken neben Variablennamen auch Klassennamen und Funktionsnamen geändert und sinnentfremdet werden können. Doch schon durch solche Veränderungen schlägt hier beispielsweise der oben aufgeführte Ansatz zum Überladen einer bestimmten Klasse fehl. Denn die Suche nach dem Klassennamen CertifikatePinner liefe dadurch ins Leere. Allerdings kann sich der Angreifer auch bei einer derart obfuskierten App zunutze machen, dass eine RFC-konforme Umsetzung eines Zertifikats-Pinning allein die Verwendung der Hashfunktion SHA-265 vorsieht, so wie wir es in Abschn. 8.1.3 bereits erfahren haben. Eine derartige RFC-konforme Umsetzung ist auch für die durch die okhttp-Bibliothek eingebundene Funktionalität gegeben. Doch gerade solch eine standard-konforme Umsetzung des Zertifikats-Pinning kann sich nun leicht als Bumerang erweisen. Dann kann der Angreifer, beispielsweise indem er mit dem Befehl grep nach SHA-256 sucht, den Bereich innerhalb der Instruktionsfolge der App herausfinden, der diese Hashfunktion verwendet, und dadurch die Funktion identifizieren, die mit hoher Wahrscheinlichkeit der Klasse CertifikatePinner zuzuordnen ist. Ist die Klasse auf diese Weise identifiziert, kann die eigentliche Funktionalität unter Einsatz des ClassLoader-Konzeptes wieder überschieben werden, so wie wir es bereits zum Aufbrechen einer gesicherten Verbindung mit Zertifikats-Pinning ohne

Obfuskierung erörtert haben. Natürlich kann man nun argumentieren, dass dann doch bitteschön die Maßnahmen der Verschleierung auch für den String SHA-256 hätte vorgenommen werden können. Dies ist richtig, zeigt aber auch nur, dass aktuell ein unentwegter Wettkampf zwischen dem Absichern und dem Hacken von Applikationen und Systemen vorherrscht.

9.1.5 Maßnahmen gegen das Aufbrechen von Zertifikats-Pinning

Dem Leser wird spätestens an dieser Stelle bewusst geworden sein, dass es eigentlich keine Möglichkeit gibt, den ausführbaren Code einer App so zu schützen, dass dieser gegen jede erdenkbare manuelle und dynamische Analyse zum Aufbrechen von Zertifikats-Pinning abgesichert wäre. Es kann also bestenfalls angestrebt werden, den zeitlichen Aufwand eines Angreifers bei der Analyse einer App deutlich zu erhöhen. Hierzu bieten sich eine Reihe von weiteren Maßnahmen an, die bei der Implementierung einer App umgesetzt werden könnten. Vier dieser Maßnahmen hat Winterer [5] im Rahmen seiner Arbeit aufgeführt:

1. *Prüfung der Code-Integrität*
   Hierbei wird zur Laufzeit einer App unter Verwendung zyklischer Redundanzprüfung (engl. *Cyclic Redundancy Check* (CRC)) oder mithilfe einer Hashfunktion eine Integritätsprüfung des auszuführenden Bytecodes und der zur Ausführung gelangenden nativen Bibliotheken vorgenommen, beispielsweise unter Verwendung der Methode crcTest(). Überprüft wird hierbei vorab, inwieweit der aktuell berechnete CRC-Wert mit dem CRC-Wert übereinstimmt, der bei der Auslieferung der App berechnet wurde. Zur Prüfung gelangen dann die CRC-Summen über alle .dex-Dateien der Applikation. Einen augenscheinlichen Wehrmutstropfen hat dieser Ansatz jedoch: Ein Angreifer dürfte natürlich auch Funktionen wie crcTest() manipulieren können, sodass diese fälschlicherweise die CRC-Prüfung mit einem positiven Ausgang beenden.
2. *Erkennung von Frida*
   Eine weitere wesentliche Schutzmaßnahme besteht darin, zur Ausführungszeit einer App zu erkennen, inwieweit auf dem System Analysewerkzeuge wie das bereits erwähnte Werkzeug Frida gestartet wurden. Die Umsetzung derartiger Erkennungsmaßnahmen kann viele Facetten haben. So kann das Absetzen des Aufrufs ps -a und das Prüfen, ob ein Prozess mit dem Namen frida gestartet wurde, bereits als sehr einfache, jedoch bei geübten Angreifern wirkungslose Umsetzung einer derartigen Schutzmaßnahme gelten. Auch die Überprüfung. über welchen TCP-Port Frida kommuniziert, kann ein geübter Angreifer verheimlichen, indem er vorab den Standard-Port ändert. Natürlich lassen sich die hier skizzierten Maßnahmen auch zur Erkennung weiterer Analysewerkzeuge anpassen.

3. *Erkennung von Root*
   Für viele der geschilderten Maßnahmen die ein Angreifer durchführt, sind Root-Rechte erforderlich. Daher ist als eine weitere Maßnahme gegen das Aufbrechen von Zertifikats-Pinning die Erkennung von vergebenen Root-Rechten zu nennen. Einige hiervon haben wir bereits genannt, als wir uns in Abschn. 8.1.3 mit den Sicherheitsvorgaben für TAN-Apps beschäftigt haben. Darüber hinaus schlägt die OWASP vor, die Aufrufe ausgezeichneter Prozesse, sowie Lese- und Schreibrechte zu prüfen. Da mit dem Werkzeug Magisk jedoch eben solche Verfahren zur Root-Erkennung verschleiert werden können, ist es darüber hinaus auch zwingend geboten, Maßnahmen zu implementieren, mit denen dieses Werkzeug zu entdecken wäre. Dies ist allerdings deutlich leichter gesagt als getan, denn Magisk bietet die Möglichkeit, mit jedem Neustart unter einem anderen Namen in der Prozessliste zu erscheinen [5].
4. *Integritätsprüfungen zur Laufzeit*
   Das Android-Betriebssystem bietet je nach Version die Funktionen GET_SIGNATURES und GET_SIGNING_SIGNATURES, mit denen es möglich ist, zur Ausführungszeit einer App eine Signaturprüfung des ausführbaren Codes durchzuführen. Damit kann die App zur Laufzeit prüfen, ob Manipulationen am Code vorgenommen wurden.

Ob die oben geschilderten Gegenmaßnahmen, die allesamt bestenfalls zur Erhöhung der zeitlichen Hürden eines Angreifers dienen können, die Verwendung von beispielsweise TAN-Apps rechtfertigen, wird der Leser sicherlich nun selbst entscheiden können.

9.2 Zusammenfassung

Wir haben zu Beginn dieses Kapitels drei Varianten einer Realisierung von Zertifikats-Pinning für Apps kennen gelernt, mit denen der Entwickler auf der Grundlage der erörterten Sicherheitsannahme „Trust-on-First-Use“ eine sichere Kopplung der abgesicherten Verbindung an das verwendete Zertifikat aufsetzen kann. Dabei dürften die Verwendung von bereits vorhandenen Bibliotheken mit Funktionen zum Zertifikats-Pinning sowie die Möglichkeit der Einstellung innerhalb der Netzwerk-Konfigurationsdatei der Option der Eigenimplementierung für die meisten App-Entwickler deutlich vorzuziehen sein. Des Weiteren hat der Entwickler einer App neben der Frage, ob ein Back-up Zertifikat zum Einsatz kommt, ebenfalls zu entscheiden, welcher Zertifikatstyp eigentlich eingesetzt wird. Die Wahlmöglichkeiten, ob Blattzertifikate, Zwischenzertifikate, oder Wurzelzertifikate verwendet werden, resultiert augenscheinlich aus dem Zielkonflikt der beiden teilweise konkurrierenden Ziele Sicherheit und Verfügbarkeit, mit denen sich ein Entwickler konfrontiert sieht. Ebenso kann auch in dem Moment, in dem die Gültigkeit eines Zertifikats abgelaufen ist, die auf diese Weise zusätzlich abgesicherte Verbindung zwischen der Smartphone-App zum jeweiligen Server nicht mehr aufgebaut werden. Gleichzeitig haben wir im Verlauf dieses Kapitels erfahren, dass Angreifer durchaus in der Lage sind, mittels Zertifikats-Pinning gesicherte Verbindungen aufzubrechen. Dies gilt

sowohl für nicht obfuskierte als auch für obfuskierte Apps. So ist es bei nicht obfuskierten Apps Mallorys Ziel, den Trust-Manager zu unterlaufen, entweder durch die Erstellung eines eigenen Trust-Managers oder aber indem die Funktionalität des gegenwärtigen Trust-Managers auf null gesetzt wird. Darüber hinaus bietet sich für Mallory das Überladen der Klassen und Funktionen, die für die Zertifikats-Pinning-Funktionalität verwendet werden, als eine weitere Möglichkeit zum Aufbrechen der Verbindung an. Hierbei verwendet der Angreifer das von Java bereitgestellte Konzept zum Überladen von Klassen, mit dem es möglich ist, zur Ausführungszeit die Funktionalität einzelner Klassen oder Methoden zu überlagern. Die Maßnahme der Obfuskierung des Programmcodes einer App mit dem Ziel, denjenigen Code der App, der für das Zertifikats-Pinning eingesetzt wird, für Mallory unerkennbar zu machen, kann wirklich nur als eine weitere Hürde für Mallory angesehen werden und darf nicht als deren vollständig ausgereifter Schutz verstanden werden. Ähnlich zu bewerten sind auch die weiteren Maßnahmen, die ein Entwickler einsetzen kann, um das Aufbrechen von Verbindungen zu erschweren. So haben wir zum Ende dieses Kapitels die vier essenziellen Maßnahmen Prüfung der Code-Integrität, Erkennung von Root, Erkennung von Analysewerkzeugen sowie Integritätsprüfungen zur Laufzeit erörtert. Abschließend möchte ich noch darauf hinweisen, dass für das hier geschilderte Aufbrechen von Verbindungen die Angreiferkategorie ‚Mallory' nicht ganz zutreffend ist. So haben wir im Grundlagenteil diese Buches Mallory zwar als aktiven Angreifer eingeführt. Allerdings würde sich die dort getroffene Kategorisierung des Angreifers Mallory ja ausschließlich mit der Manipulation von Nachrichten auf der Übertragungsstrecke beschäftigen. Die hier geschilderten Manipulationen nimmt unser Angreifer jedoch an einem Endpunkt der gesicherten Verbindung vor, also an Software auf dem Gerät. Dies geschieht allerdings mit dem übergeordneten Ziel, dass sich diese Manipulationen dann wiederum auf die Nachrichten auswirken, die von diesem Gerät versendet werden, um sie auf diese Weise auf der Übertragungsstrecke abhören zu können. Und noch ein weiterer Aspekt, der nicht nur die – vielleicht noch zu belächelnde – akademische Einordnung des Angreifers bezüglich seiner Namensgebung betrifft, sondern die Brisanz der tatsächlichen praktischen Auswirkungen eines solchen Angriffs dokumentiert: Da die hier geschilderten Manipulationen nun Auswirkungen auf die Absicherung höherer Protokollschichten wie der Transportschicht haben, muss sich der Angreifer nach dem Aufbrechen der Sitzung zum eigentlichen Abhören nicht zwingend innerhalb der drahtlosen Übertragungsreichweite auf dem ersten Sprung (engl. *hop*) zwischen Smartphone und Basisstation befinden. Vielmehr ist es aus der Perspektive des Angreifers vorteilhaft, die Nachrichten an irgendeinem Netzkoppelelement auf der weiteren Übertragungsstrecke zwischen Smartphone und Server abzuzweigen. Für eine auf einem Smartphone laufende mobile Banking-Anwendung kann der Angreifer nun beispielsweise die auf dem Smartphone zu verwendenden TANs mitlesen, obwohl die Anwendung der Sicherheitsvorgabe einer technischen Umsetzung von Zertifikats-Pinning unterliegt, und zwar unabhängig davon ob separate Apps oder eine einzige App für TAN-Empfang und Banking verwendet werden. Ein ähnliches Abzweigen sensibler Daten oder brisanter Informationen ist natürlich auch durch das Aufbrechen einer Verbindung bei anderen Apps zu beobachten.

Literatur

1. Cipolloni, P.: Universal Android SSL pinning bypass with Frida. https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/. Zugegriffen: 11. Juli 2019
2. Okhttp: https://square.github.io/okhttp/. Zugegriffen: 14. Aug. 2019
3. Sensepost: Attempts to bypass SSL pinning implementations. https://github.com/sensepost/objection/blob/master/agent/src/android/pinning.ts. Zugegriffen: 31. Okt. 2019
4. Vinci, M.: Universal Android SSL pinning bypass. https://techblog.mediaservice.net/2018/11/universal-android-ssl-pinning-bypass-2/ (2018). Zugegriffen: 31. Okt. 2019
5. Winterer, U.: Android Security – Umgehen von Certificate Pinning unter Verwendung gängiger Tools in aktuellen Versionen ab Android 7 (Nougat) – API Level 24. Masterarbeit in ENITS, Hochschule Offenburg (2019)

Obfuskierung und Deobfuskiereung 10

10.1 Techniken zur Obfuskierung und Deobfuskierung von Apps

Das Erstellen manipulationssicherer mobiler Anwendungen wie der im vorhergehenden Abschnitt beschriebenen Banking-Apps wird noch dadurch erschwert, dass mittlerweile ein ganzes Arsenal an Analysewerkzeugen zum Reverse Engineering von Apps verfügbar ist. Beim Reversing von Android-Applikationen möchte man – ausgehend von dem auf dem mobilen Gerät zwangsläufig verfügbaren ausführbaren Code – möglichst genaue Schlüsse auf die Beschaffenheit des Quellcodes einer App ziehen. So kann es wahlweise einen Sicherheitsexperten oder aber Angreifer interessieren, ob und wenn ja welche IP-Adressen bzw. URI Daten die App verlassen oder inwieweit sensible Informationen wie Schlüsselmaterial fest im Quellcode der App kodiert sind. Es könnte neben den genannten Gründen ja auch durchaus denkbar sein, dass der Code Funktionalität enthält, die nur unter ganz bestimmten Vorbedingungen zur Ausführung gelangt. Derartige Techniken zur Deobfuskierung der Beschaffenheit und Funktionalität von Apps, lassen sich einteilen in solche, die auf einer rein statischen Codeanalyse beruhen, und solche, die mittels dynamischer Codeanalyse das Verhalten einer App zur Laufzeit beobachten. Daneben haben sich auch Techniken zur Obfuskierung von Daten sowie der Funktionalität einer App etabliert. Hierbei geht es im Wesentlichen darum, den Programmcode so abzuändern, dass noch immer die geplante Funktionalität ausgeführt wird, es aber für Personen die die App einem Reverse Engineering unterziehen deutlich schwerer wird Rückschlüsse auf den originalen lesbaren Quellcode und damit auf eventuell verborgene Funktionalitäten zu ziehen.

Solche Obfuskierungstechniken sind sicherlich für Angreifer zur Verschleierung von Schadsoftware und Hintertüren von großem Interesse. Sie können aber auch dazu dienen, das geistige Eigentum des Entwicklers zu schützen.

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_10

Darüber hinaus können diese Techniken jedoch auch wertvoll sein bei der herkömmlichen App-Entwicklung. Hiermit wäre es beispielsweise möglich, die konkrete Umsetzung der Sicherheitsvorgaben von TAN-Apps zu verheimlichen. Beispielsweise könnten auf diese Weise die verwendeten Verfahren zur Root-Erkennung verschleiert werden. Noch eine Anmerkung zu den Vorzügen von Verfahren der statischen und der dynamischen Codeanalyse. Sicherlich ist die dynamische Codeanalyse sehr mächtig. Allerdings wird man nur schwerlich ganz ohne statische Codeanalyse auskommen können. Denn eine Reihe von Apps prüfen zur Laufzeit, ob sie aktuell innerhalb eines Emulators zur dynamischen Codeanalyse ausgeführt werden. Ist dies der Fall, verändern sie dementsprechend ihr Verhalten. Solche Fallunterscheidungen lassen sich oftmals deutlich besser mithilfe einer statischen Codeanalyse erkennen.

Doch der Reihe nach: Wie wir innerhalb von Abschn. 7.1 bereits erörtert haben, ist jede App zur Laufzeit ein eigener Prozess der Android Runtime (ART). Zur Ausführung gelangt hierbei dex-Bytecode, wobei die eigentliche Implementierung des Entwicklers in Java erfolgte. Damit die App auf dem Zielgerät ausführbar ist, muss diese als Datei mit der Endung .apk vorliegen. Dieses Paket enthält Bilder, Zugriffsrechte, Bytecode sowie Informationen zum Layout. Der eigentliche Dalvik-Bytecode, die dex-Datei, ist dabei ähnlich strukturiert wie Java-Bytecode. Jede App wird mit einem Schlüssel des Entwicklers signiert wobei Apps, die mit dem gleichen Entwicklerschlüssel signiert sind, auch unter der gleichen UID zur Ausführung gelangen.

10.1.1 Techniken zur Deobfuskierung

10.1.1.1 Werkzeuge zur statischen Codeanalyse

Die folgende Auswahl von Werkzeugen bieten sich zur statischen Analyse einer Android-App in einer Datei mit der Endung .apk an. Um diese Werkzeuge einsetzen zu können, sollte die App vorab beispielsweise mit unzip entpackt sein und in extrahierte apk-Dateien überführt worden sein. Insbesondere die Datei classes.dex liegt nun vor. Als Werkzeuge, mit denen eine statische Codeanalyse einer App vorgenommen werden kann, bieten sich an:

1. **JD-GUI:** Dies ist ein grafischer Decompiler, mit dem Java-Bytecode in eine Quellcode- Darstellung wie Java-Quellcode, Jimple oder smali überführt werden kann.
2. **dex2jar:** Konvertiert eine App mit der Endung der .apk in eine Java-JAR-Datei. Das Dalvik Executable Format (.dex/.odex) kann in den Java-Bytecode überführt werden. Der Aufruf `d2j-dex2jar.sh classes.dex` erzeugt eine Datei `classes_dex2jar.jar`. Danach ist die jar-Datei, also der Java-Bytecode, mit dem grafischen Decompiler JD-GUI analysierbar.
3. **baksmali/smali:** Die Komponente baksmali ist ein Disassembler, also wenn man so will, ein entgegengesetzter Assembler, für Dateien, die im .dex-Format vorliegen. Das Ergebnis ist eine Mischung aus Pseudocode und Java. Dieser so entstandene Code

kann daraufhin verändert werden und ist anschließend mittels der Komponente smali wieder zurückführbar in das .dex-Format. Es können also Veränderungen am Code getätigt werden, ohne dass hierzu der eigentliche Source-Code verfügbar sein muss.

4. **IDA Pro:** IDA Pro ist ein interaktiver Disassembler und Debugger. Der Assembler-Prozess kann über eine Disassemblierung rückgängig gemacht werden, sodass aus dem nativen Maschinencode wieder Assembler-Code gewonnen wird. Für Android kann er zur Bytecode-Analyse sowie zur Analyse nativen Codes genutzt werden. Das von Hex-Rays entwickelte Werkzeug ist als Free Edition und als Vollversion erhältlich. Die freie Version beschränkt sich jedoch auf die Analyse von Code für die x86-Architektur.
5. **APKtool:** Mit diesem Werkzeug lassen sich der Aufbau einer App und seine Strukturierung nachvollziehen. Es lassen sich Ressourcen lesen und modifizieren, wie beispielsweise die Manifest-Datei *AndroidManifest.xml*. Allerdings ist es mit dem APKtool auch möglich eine apk-Datei zu dekompilieren, den Code der App zu verändern und anschließend die so geänderte App wieder zu kompilieren und zur Ausführung zu bringen. Über diesen Weg könnte ein Angreifer beispielsweise versuchen, die von den Entwicklern implementierte Root-Erkennung einer TAN-App zu unterbinden.

Neben diesen schon seit längerer Zeit verfügbaren Werkzeugen zur statischen Codeanalyse von Apps sind in der jüngsten Zeit eine Reihe von Werkzeugen aus Forschungsaktivitäten verschiedener Gruppen entstanden: Werkzeuge wie Amandroid [9], DroidSafe [4], FlowDroid [1] oder IccTA-K3 [5] dienen ebenfalls der statischen Analyse von Android-Apps. Allerdings lassen diese eine aussagekräftige Datenflussanalyse von Apps mit eingebundenem nativem Code nur unzureichend zu. Konkret bedarf es hierfür einer integrierten und aussagekräftigen Datenflussanalyse von Dalvik-Bytecode und nativen Binaries. Denn mit dem Native Development Kit (NDK) ist es einem App-Entwickler jederzeit möglich, nativen Code in Apps einzubinden. Dabei dient ihm das Java Native Interface (JNI) als Schnittstelle zur Kommunikation zwischen Java-Methoden und nativen Funktionen. Die Verwendung solch einer Schnittstelle allein ist jedoch noch lange kein Indiz für Malware, denn häufig wird nativer Code ganz legitim verwendet, beispielsweise zur Steigerung der Performanz bei rechenintensiven Aufgaben. Das im Jahre 2018 von den Sicherheitsexperten Fengguo Wei und Co-Autoren vorgestellte Werkzeug JN-SAF [10] ermöglicht nun auch die statische Analyse von Apps, die nativen Code wie C oder C++ enthalten (siehe Abschn. 10.1.2). Hierzu kommt ein Algorithmus mit Namen *Summary-based Bottom up Dataflow Analysis* (SBDA) zum Einsatz. SBDA benötigt als wesentliche Eingabe einen Graphen der aufrufenden Routinen (engl. *call graph*) und berechnet daraus auf der Grundlage eines topologischen Sortieralgorithmus (in umgekehrter Reihenfolge) eine Liste mit Methodenaufrufen so, dass die aufgerufene Methode immer vor der sie aufrufenden Methode aufgeführt ist. Eventuelle Schleifen im Call-Graphen werden dabei so behandelt, dass jede Methode nur genau einmal in der resultierenden Liste steht. Auf der Grundlage der so erzeugten Ergebnisse der Datenflussanalyse wird dann ein Analysebericht erstellt. In [10] führen die Entwickler von

JN-SAF eine Reihe von Teilproblemen auf, die eine integrierte statische Datenflussanalyse aufwendig und fehlerträchtig machen: So basieren die statischen Analysewerkzeuge beider Welten auf verschiedenen Datenrepräsentationen. Des Weiteren ist der Einsatz existierender Analysewerkzeuge für Binaries zwar möglich, er bedarf jedoch der Harmonisierung. Weitere Fallstricke solch einer integrierten statischen Datenflussanalyse sind das Auflösen nativer Methodenaufrufe und die Handhabung nativer Aktivitäten. Auch wenn Wei und Co-Autoren ihre Ergebnisse in [10] mit ausgiebigen Vergleichsläufen (engl. *benchmarks*) unterfüttern, so bleibt es insgesamt doch recht undurchsichtig wie aussagekräftig ein mit JN-SAF erzeugter Analysebericht wirklich ist.

Es ist Ihnen sicherlich aufgefallen, dass sich die Funktionalitäten der genannten Analysewerkzeuge in Teilbereichen überschneiden. So sollte jeweils von Fall zu Fall entschieden werden, welche dieser Werkzeuge tatsächlich zur Anwendung gelangen und für das aktuelle Reversing-Projekt am vorteilhaftesten sind. Sicherlich ist es immer auch von Bedeutung, mit welchen dieser Werkzeuge der Pentester bereits im Vorfeld Erfahrungen sammeln konnte. Ähnliche Werkzeuge zur statischen und dynamischen Analyse von Apps lassen sich auch für iOS und Windows Mobile nennen. Einen Überblick hierzu hat Michael Spreitzenbarth in [8] zusammengetragen.

▶ Werkzeuge zur statischen Codeanalyse von Apps mit eingebundenem nativem Code liefern oft nur unzureichende Analyseergebnisse. Diese integrierte statische Datenflussanalyse ist aufwendig und fehlerträchtig, da i) die statischen Analysewerkzeuge beider Welten auf verschiedenen Datenrepräsentationen basieren, ii) existierende Analysewerkzeuge für Binaries zwar eingesetzt, jedoch harmonisiert werden müssen und iii) das Auflösen nativer Methodenaufrufe und die Handhabung nativer Aktivitäten zusätzliche Anforderungen stellen.

10.1.1.2 Werkzeuge zur dynamischen Codeanalyse

Eine automatisierte und dynamische Codeanalyse lässt sich beispielsweise mit dem Analysewerkzeug DroidBox durchführen. Hierzu ist ein Emulator mit entsprechender Konfiguration zu verwenden, um zum einen die Android-Architektur auf dem Zielsystem nachzubilden und zum anderen die zu untersuchende App in einer abgeschotteten Umgebung laufen zu lassen. Aktiviert man die Python-Skripte in DroidBox unter Angabe der zu analysierenden App, wird der Bericht (engl. *report*) hierzu in ein JSON-Verzeichnis geschrieben und ist anschließend auswertbar.

Will man eine App zur Laufzeit analysieren, indem man einzelne Belegungen von Variablen ändert, um beispielsweise den Programmfluss zu ändern, so bieten sich Werkzeuge wie Codeinspect oder einfach ein Debugger an, mit dem man Haltepunkte (engl. *breakpoints*) setzen und Variablenbelegungen ändern kann. Hierdurch lässt sich der Prozess einer dynamischen Codeanalyse deutlich zielführender vorantreiben. Allerdings muss der ursprüngliche Entwickler hierzu im Manifest der App

```
android:debuggable="true"
```

gesetzt haben. Andernfalls ist eine dynamische Codeanalyse unter Verwendung eines Debuggers zunächst einmal nicht möglich. Mithilfe der oben bereits erwähnten Komponenten baksmali und smali sollte die jedoch vorab enthaltene Einstellung

```
android:debuggable="false"
```

änderbar sein, sodass daraufhin die Codeanalyse mittels Debugger wieder möglich ist.

Eine potenziell bösartige App sollte zur dynamischen Analyse zwingend innerhalb einer isolierten Umgebung wie beispielsweise der cuckoo Sandbox [3] gestartet werden. Oftmals muss für die dynamische Codeanalyse jedoch noch ein ‚Fake'-Server erstellt werden, damit der potenziellen Malware eine Interaktion mit dem heimischen Server vorgegaukelt werden kann für den Fall, dass sie Passagen enthält, in denen sie auf eingehende Nachrichten eines bestimmten Servers wartet oder Nachrichten dorthin senden möchte. Das erste Ziel bei einer strukturierten Analyse einer App sollte es nun sein, sich einen Überblick über den Kontrollfluss zu verschaffen. Der Kontrollflussgraph einer potenziellen Malware kann beispielsweise mit dem frei verfügbaren Werkzeug GroddDroid [2] erzeugt werden. Solch ein Graph setzt sich zusammen aus einer Menge von Knoten, die die Grundblöcke des Programms darstellen, sowie einer Menge gerichteter Kanten, die die Übergänge bzw. Programmabläufe zwischen diesen Grundblöcken abbilden. Mit der Erstellung des Kontrollflussgraphen für die zu analysierende App weist das Tool jedem einzelnen Methodenaufruf einen Risikowert zu mit dem Ziel, diejenigen Passagen im Kontrollfluss mit den höchsten Risikowerten zu identifizieren. Die Überlegung der Sicherheitsexperten ist dabei die, dass über diese Wertevergabe (engl. *scoring*) hauptverdächtige Passagen im Bytecode einer potenziell bösartigen App identifizierbar sind. Dies ist allerdings nur der erste Schritt. Typischerweise ist es so, dass Malware wie beispielsweise die bekannten Vertreter SaveME, WipeLocker oder DroidKungFu ihren bösartigen Teil unterschiedlich aktivieren. Um die verdächtige Passage im Code innerhalb der kontrollierten Umgebung auszuführen, ist es also notwendig, das Ereignis, das ihre Aktivierung auslöst, nachzuvollziehen. Mal erfolgt dies unmittelbar mit dem Starten der App, mal nach einem Reboot oder die App wartet erst einmal eine vordefinierte Zeit, bevor tatsächlich die bösartige Passage zur Ausführung gelangt. Erschwerend kommt hinzu, dass mehr und mehr schadhafte Apps erkennen, wenn sie in einer kontrollierten Umgebung laufen. Dann werden sie die auslösende Bedingung (engl. *triggering condition*) nicht umsetzen, mit der sie in die Passage mit der schadhaften Codesequenz eintreten. Werkzeuge wie GroddDroid ändern daher wenn notwendig den Bytecode der potenziellen Malware gezielt so um, dass Variablenbelegungen die auslösende Bedingung für den Eintritt in eine Passage des Codes mit sehr hohem Risikowert erzwingen. Jetzt kann die eigentliche Analyse der potenziell bösartigen Passage erfolgen. Denn eine Passage, die mit einem hohen Risikowert belegt wurde, muss nicht zwangsläufig auch Schadcode enthalten. Umgekehrt gilt aber leider auch, dass Abschnitte im Kontrollflussgraphen, die einen niedrigen Risikowert erhalten haben,

nicht zwangsläufig frei von Schadcode sind. So bleibt festzuhalten, dass Werkzeuge, die die einzelnen Abschnitte des Kontrollflussgraphen einer App mit Risikowerten belegen, sicherlich zu einem enormen zeitlichen Gewinn bei der dynamischen Analyse von Apps beitragen, und dies insbesondere im Vergleich zu Werkzeugen (Gorilla), die eher ‚stumpf' die Eingaben eines Nutzers simulieren. Zu nennen sind hier die Werkzeuge *The Monkey, PuppetDroid,* oder *Android Automatic App Explorer.* Nach wie vor werden also genügend schadhafte Apps eine derartige dynamische Analyse passieren, ohne dass diese zwangsläufig als Malware eingestuft werden.

- Werkzeuge zur dynamischen Codeanalyse von Apps erzeugen häufig einen gerichteten Kontrollflussgraphen, der mit Risikowerten für einzelne Methodenaufrufe versehen ist. Für diejenigen Passagen, die auf diese Weise die höchsten Risikowerte erhalten haben, wird anschließend eine Aktivierung der auslösenden Bedingung angestrebt, die den Eintritt in die möglicherweise Schadcode enthaltende Passage erzwingt. Allerdings müssen Passagen mit einem hohen Risikowert nicht zwangsläufig auch Schadcode enthalten. Umgekehrt müssen Abschnitte im Kontrollflussgraphen mit niedrigen Risikowerten nicht zwangsläufig frei von Schadcode sein.

10.1.2 Einfache Techniken der Bytecode-Obfuskierung

Um wahlweise den Sicherheitsexperten oder aber einem Angreifer die statische sowie die dynamische Codeanalyse mit den oben genannten Werkzeugen zu erschweren, lassen sich eine Reihe von Techniken einsetzen, mit denen der Bytecode einer mobilen Anwendung obfuskiert werden kann. Er soll also so verschleiert werden, dass eine Analyse seiner vollumfänglichen Funktionalität sehr erschwert wird oder im Idealfall gar nicht mehr möglich ist. Wir begnügen uns an dieser Stelle damit, sehr einfache Techniken aufzuführen, mit denen dies erreicht wird oder zumindest angestrebt werden kann. Die hier aufgeführten Techniken wurden mehrheitlich von Moritz Kaumanns genauer untersucht und in [6] festgehalten.

10.1.2.1 Obfuskierung von Metadaten

Statische Zeichenketten sollten tunlichst keine Rückschlüsse auf die Funktionalität von Klassen oder Methoden zulassen. Um dies zu umgehen werden Zeichenketten durch eine im Bytecode enthaltene Funktion zur Laufzeit umgewandelt. Hierzu ist es allerdings erforderlich den eigentlichen Text zur Entwicklungszeit zu verschleiern. Im ausführbaren Bytecode der App muss also eine Funktionalität enthalten sein, nennen wir sie $D(\,)$ sodass

$$m = D(m_o)$$

für alle m aus einer endlichen Menge von Zeichenketten. Vorab hat der Entwickler der App eine Funktion $O(\,)$ verwendet mit der er die benötigten ursprünglichen Zeichenketten m in ihre obfuskierten Varianten überführt:

$$m_o = O(m)$$

Nur die Funktion $D(\,)$ ist Bestandteil des ausführbaren Codes der App. Sie kann als eine einzige Methode umgesetzt oder gestückelt sein und sich über den gesamten ausführbaren Code erstrecken. Natürlich kann ein halbwegs versierter Sicherheitsexperte mittels statischer Codeanalyse dennoch durch eigenes Verwenden von $D(\,)$ auf die Zeichenkette m folgern. Der Analyseaufwand lässt sich aber schon dann bereits beachtlich erhöhen, wenn beispielsweise verschiedene Parameter für verschiedene Zeichenketten eingesetzt werden. Dies dokumentiert, dass es für die Umsetzung von $D(\,)$ verschiedene Spielarten gibt, wobei die prinzipielle Herangehensweise jedoch immer die gleiche ist.

Diese für sich genommen recht banale Technik kann nun mit weiteren Techniken zur Verschleierung der Funktionalität des Codes kombiniert werden, beispielsweise durch die Umbenennung von Identifizierern für Klassennamen, Methodennamen, Variablennamen oder aber auch den Namen genutzter Pakete. Mit dem Werkzeug ProGuard und anderen vergleichbaren Werkzeugen wie JavaGuard, DashO oder Zelix KlassMaster existieren bereits Werkzeuge, die neben einer Komprimierung auch derartige Obfuskierungstechniken automatisiert einsetzen. Dabei wird bei Verwendung von ProGuard, das im Build-System von Android integriert ist, ein voreingestelltes Abbild aller Zeichenketten und Identifizierer in der Datei *mapping.txt* abgelegt. Es ist offensichtlich, dass diese danach auf jeden Fall angepasst werden sollte. Eine schematische Arbeitsweise von Obfuskierern, so wie dies mittels ProGuard geschieht, ist in Abb. 10.1 dargestellt. Die Klasse GeheimnisLesen, ihre Methoden sowie Variablennamen werden nach einem Schema obfuskiert, wie es in der Datei *mapping.txt* festgelegt ist.

Gemeinsam mit den oben beschriebenen Techniken zur Verheimlichung von Zeichenketten wird auf diese Weise der statische Analyseaufwand bereits deutlich erhöht.

Neben diesen doch noch recht einfachen Techniken haben sich auch Techniken wie das Einfügen von Junk-Bytes etabliert. Hierbei werden weitere Instruktionen in den Dalvik-Bytecode (dex-Datei) eingeführt, ohne jedoch die ursprünglich intendierte Programmabfolge hierdurch zu verändern. Dies erfolgt über das Einfügen von Sprunganweisungen. Allerdings ist das statische Disassembling von derartigen nachträglich eingefügten Sprunganweisungen sehr wohl betroffen. Der Sprung wird als normaler Sprung aufgefasst. Jedoch werden die nachfolgenden Bytes, die eigentlich übersprungen werden sollten, weiter übersetzt. Diese Vorgehensweise dient dazu, Übersetzungen zu manipulieren. Der Sicherheitsexperte Patrick Schulz hat diese Thematik genauer analysiert [7].

10.1.2.2 Dynamisches Laden von Programmcode

Eine Codeanalyse ist immer dann besonders schwer durchzuführen, wenn Teile des ausführbaren Programms zum Zeitpunkt des Startens der App noch gar nicht vorliegen, sondern erst zur Laufzeit angefordert werden. Dann gelangen diese Codesequenzen jedoch direkt zur Ausführung. Für Android-Apps kann dies auf mehrere Arten geschehen: Zum einen kann weiterer Bytecode eingebunden werden. Zum anderen ist es möglich, dass nativer Code wie C oder C++ zur Ausführung gelangt. Und schließlich gibt es über

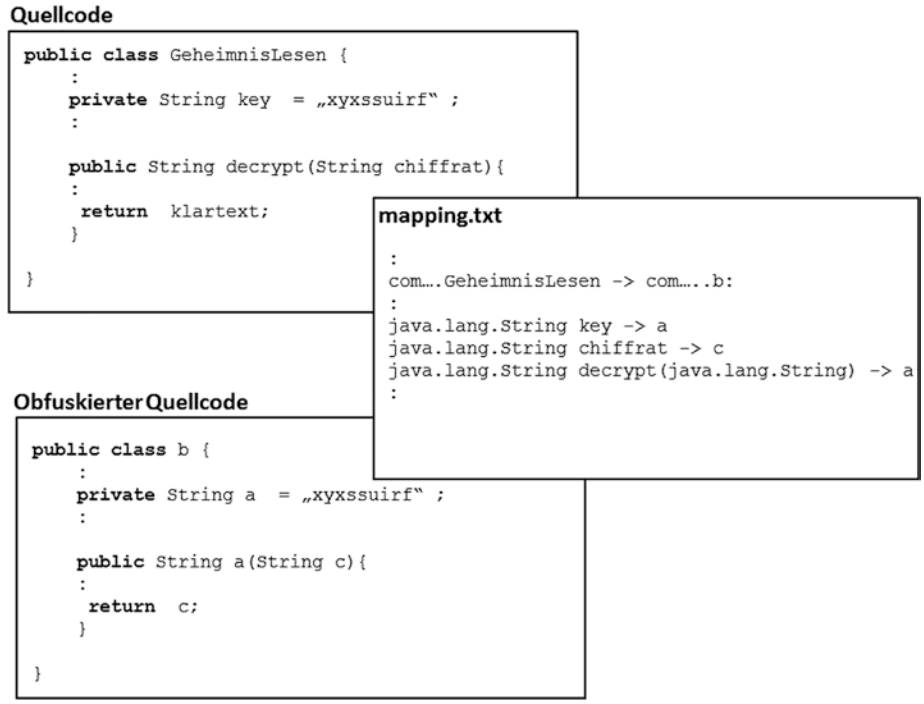

Abb. 10.1 Schematische Darstellung einer Java-Quelltext-Obfuskierung mithilfe von Werkzeugen wie ProGuard. (Quelle: eigene Darstellung in Anlehnung an [6])

das Einbinden von Webseiten direkt in die Applikation die Möglichkeit, JavaScript-Code zu interpretieren. Jeder dieser Mechanismen ist, wie man sich vorstellen kann, für einen Entwickler schadhafter Applikationen hoch attraktiv.

Java bzw. Android-Apps können zur Laufzeit einer App ganze Klassen nachträglich einbinden. Für Android ist hierzu der DexClassLoader zuständig. Er ermöglicht das Nachladen einzelner Klassen von der .apk-Datei oder Java-Bytecode, also Java-jar. Deutet also der Quellcode einer App darauf hin, dass ein ClassLoader eingesetzt wird, ist Vorsicht geboten. Dies könnte dann in seiner einfachsten Variante wie folgt aussehen:

```
public class MainActivity extends AppCompatActivity {
 :
 DexClassLoader classloader = new DexClassLoader(…, …, this.getClass().
getClassLoader());
 :
 Class run = classloader.loadClass("…..");
 :
}
```

Natürlich können diese Instruktionsfolgen auch wieder mit ProGuard oder anderen Verfahren zur Umbenennung von Identifizierern für Klassennamen, Methodennamen, Variablennamen bzw. genutzter Pakete bearbeitet worden sein, um die Verwendung des DexClassLoader zu verheimlichen.

Verwendet ein Entwickler darüber hinaus das Android Native Development Kit (NDK) kann in eine Android-App auch C- bzw. C++-Code eingebunden werden. Da der nun eingebundene Code aber als Binary vorliegt, führt eine einfache Dekompilierung mittels des Java-Dekompilers bei der statischen Codeanalyse also nicht mehr zum Ziel. Es ist also die Codeanalyse von Binaries erforderlich. Doch eine derartige Binary-Analyse gestaltet sich typischerweise als deutlich aufwendiger. Zudem ist hierbei ein dediziertes Expertenwissen unabdingbar. Da der native Code im Gegensatz zum Bytecode plattformabhängig ist, muss dieser im Vorfeld für die entsprechenden Zielplattformen kompiliert vorliegen. Liegt er als x86-Code vor, kann er mithilfe der IDA Pro Free Edition analysiert werden.

WebView ist eine Engine für Android, mit der sich innerhalb einer App Web-Inhalte darstellen lassen. Über die Schnittstelle addJavaScriptInterface ist es dem Entwickler der App also möglich Javascript interpretierbar zu machen:

```
webView.addJavascriptInterface(…);
```

Eine App mit solch einer Instruktion wird bei Aufruf eines Webservers über eine http(s)-Verbindung also jeglichen in der Webseite eingebetteten JavaScript-Code zur Ausführung bringen und damit verwundbar gegenüber schadhaften Skripten sein. Ähnlich wie beim dynamischen Laden von Bytecode oder nativen Code ist auch in diesem Fall die Codeanalyse der App erschwert.

10.2 Zusammenfassung

In Rahmen dieses Kapitels haben wir einen kompakten Einblick in elementare Techniken zur Obfuskierung sowie zur Deobfuskierung von ausführbarem Code von Android-Apps erhalten. Wir haben verstanden, warum Techniken der Obfuskierung von Instruktionsfolgen nicht nur aus der Perspektive des Angreifers wertvoll sind. Darüber hinaus können sie auch bei der Entwicklung von gutartigen Apps helfen, beispielsweise indem ein Mechanismus zur Root-Erkennung gegen Gegenmaßnahmen zu härten ist oder aber das geistige Eigentum des Entwicklers auf diese Weise zu schützen wäre. Wir haben erfahren, dass bereits eine ganze Reihe an Werkzeugen zur statischen Analyse von Android-Apps verfügbar sind. Aber insbesondere die statische Codeanalyse von Apps mit eingebundenem nativem Code liefert oftmals nur Analyseergebnisse, die mit Vorsicht zu genießen sind. Denn eine derartige integrierte statische Datenflussanalyse ist aufwendig und fehlerträchtig. Darüber hinaus existieren mit den DroidbBox-Python-Skripten, Codeinspect oder einem Debugger auch etablierte Werkzeuge, die eine dynamische Codeanalyse zur Ausführungszeit einer App ermöglichen. Noch eher dem Bereich

der Forschung zuzuordnen ist der durchaus vielversprechende Ansatz GroddDroid. Hier wird als Erstes ein Kontrollflussgraph der einzelnen Methoden erzeugt, der mit Risikowerten für einzelne Methodenaufrufe versehen ist. Für Ausführungsblöcke, die auf diese Weise die höchsten Risikowerte erhalten haben, wird anschließend eine Aktivierung der auslösenden Bedingung angestrebt, um Eintritt in die möglicherweise Schadcode enthaltende Passage zu erwirken. Auch wenn solch ein Vorgehen nicht frei ist von Fehlinterpretationen, was die Gutartigkeit bzw. Schadhaftigkeit von Ausführungsblöcken betrifft, so kann das Vorgehen doch als gute erste Sichtung der App mit einem enormen Potenzial für Zeitersparnissen bewertet werden.

Neben einfachen Obfusierungstechniken wie der Verschleierung der Funktionalität des Codes, beispielsweise durch die Umbenennung von Identifizierern für Klassennamen, Methodennamen, Variablennamen oder aber auch den Namen der verwendeten Pakete, bietet das dynamische Laden von Programmcode eine weitere Möglichkeit, die eigentliche Funktionalität einer App zu verschleiern. Die aufgeführten Mechanismen, mit denen der Entwickler einer App diese in die Lage versetzt, zur Laufzeit ihre Funktionalität zu ändern oder auf vielfältige Weise zu erweitern, offenbart ein weiteres Mal die doch sehr beschränkte Handhabe und Risikofreudigkeit, mit der wir uns als Nutzer eines Smartphones mit diversen Apps konfrontiert sehen. Sind die eingesetzten Technologien wie der DexClassLoader, WebView oder das Android Native Development Kit dann selbst wieder obfuskiert, so dürfte es auch für geübte Sicherheitsexperten sehr schwer werden, die tatsächliche Funktionalität einer App aufzudecken.

Die in diesem Kapitel geschilderten Techniken zur Obfuskierung sowie zur Codeanalyse bzw. zur Deobfuskierung von Android-Apps bieten natürlich nur einen ersten Einblick in diesen Themenbereich, wohlwissend, dass es zu diesem Themenkomplex noch weitaus mehr zu erörtern gäbe. Zusammenfassend lässt sich feststellen, dass es sowohl für die Verschleierung der Instruktionsfolgen und Funktionalität einer App als auch für deren Codeanalyse nachvollziehbare Gründe gibt. Allerdings erweist sich keine der beschriebenen Techniken als ‚beweisbar sicher', so wie dies im Bereich der Kryptografie zumindest angestrebt wäre. In der hier beschriebenen Unterkategorie zu mobiler Sicherheit gilt eher das Credo: Wir machen es der Gegenseite so schwer als möglich, wohl wissend, dass es nur eine Frage der Zeit ist, bis auch die neueste technische Herausforderung gebrochen ist. Das Wissen um diese Techniken ist nichtsdestotrotz essentiell für eine solide Risikobewertung von Schwachstellen und Angriffsszenarien im Kontext mobiler digitaler Geräte.

Literatur

1. Abraham, A., Andriatsimandefitra, R., Brunelat, A., Lalande, J.-F., Viet Triem Tong, V.: GroddDroid: a gorilla for triggering malicious behaviors. IEEE 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, Puerto Rico (2015)
2. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: FlowDroid: precise Context, flow, field, objecte-sensitive and lifecycle-aware taint analysis for Android apps. ACM Programming Language Design and Implementation (PLDI). ACM Digital Libary, New York (2014)
3. Cuckoo: https://cuckoosandbox.org/. Zugegriffen: 19. Nov. 2019
4. Gordon, M.I., Kim, D., Perkins, J.H., Gilham, L., Nguyen, N., Rinard, M.C.: Information flow analysis of Android applications in DroidSafe. NDSS, San Diego (2015)
5. Kaumanns, M.: Android Applikations-Schutz, Möglichkeiten zur Erschwernis von Code-Analysen. Security Trends, Hochschule Offenburg (2016)
6. Li, L., Bartel, A., Bissyande, T.F., Klein, J., Le Traon, Y., Arzt, S., Rasthofer, S., Bodden, E., Octeau, D., McDaniel, P.: Iccta: Detecting inter-component privacy leaks in android apps. IEEE ICSE (2015)
7. Schulz, P.: Dalvik Bytecode Obfuscation on Android. https://dexlabs.org/blog/bytecode-obfuscation. Zugegriffen: 26. Febr. 2019
8. Spreitzenbarth, M.: Mobile Hacking – Ein kompakter Einstieg in Penetration Testing mobiler Applikationen – iOS, Android und Windows Mobile. dpunkt.verlag (2017)
9. Wei, F., Roy, S., Ou, X., Robby, R.: Amandroid: A precise and general inter-component data flow analysis framework for security Vetting of Android apps. ACM Conference on Computer and Communications Security (CCS). ACM Digital Libary, New York (2014)
10. Wei, F., Lin, X., Ou, X., Chen, T., Zhang, X.: JN-SAF: Precise and efficient NDK/JNI-aware inter-language analysis framework for security Vetting of Android applications with Native Code. ACM Conference on Computer and Communications Security (CCS). ACM Digital Libary, New York (2018)

QR-Codes, Web-Apps und der Air-Gap

11

11.1 QR-Codes und Web-Apps

Immer dann, wenn Apps im Wesentlichen ohne Interaktion mit dem Nutzer über einen Link oder genauer einen *Uniform Resource Locator* (URL), eine Verbindung zu einem entfernten Server erzeugen, besteht ein Sicherheitsrisiko. Der Nutzer könnte Opfer eines *Drive-by-download*-Angriffs werden. Schon durch das Aufrufen einer mit aktiven Inhalten wie JavaScript angereicherten Webseite kann das Gerät mit Schadcode infiziert werden. Vorsicht ist dabei insbesondere beim Scannen von Quick-Response-(QR-)Codes geboten, denn diese enthalten oftmals gekürzte URLs, deren adressiertes Ziel nicht wirklich ersichtlich ist. Der Nutzer, der beispielsweise mit einer Kamera-App und einer App zum Scannen von QR-Codes diesen also einliest, muss sich vollständig darauf verlassen, dass die auf diese Weise angesprochene URL einen Webserver adressiert, von dem ein HTML-Dokument geladen wird, das keinen bzw. nur gutartigen aktiven Code enthält. Denn sowie diese Seite im Browser geladen ist und zur Anzeige gelangt, sorgt die im Browser integrierte JavaScript-Engine dafür, dass der darin eingebettete aktive Code direkt zur Ausführung gelangt. Ärgerlich wäre beispielsweise ein einfacher DoS-Angriff, bei dem die Ausführung des Skriptes erst die Funktionalität des Browsers und schließlich die des gesamten Gerätes einschränkt. Sehen wir uns zur Verdeutlichung den unten aufgeführten JavaScript-Code an mit einer HTML-Seite und eingebettetem JavaScript-Code zur Ausführung einer rekursiven Berechnung der ersten Zahlen aus der Fibonacci-Zahlenreihe – natürlich hätte es für solch einen einfachen DoS-Angriff auch eine while-Schleife im Skript getan mit einer Ausführungsbedingung, die immer auf wahr gesetzt ist.

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_11

```
<html>
<head><title>Fibonacci Test Page</title>
</head>
<body>
<h1> The Fibonacci Series</h1>
<script>
   function fibonacci(n)
   {
    if (n>1) return fibonacci(n-1)+fibonacci(n-2);
    if (n>=0) return 0;
    return 1;
   }
    for(i=0;i<100000;i++){
    document.write("Fibonacci Number "+i+" is "+fibonacci(i)+"<br>");
   }
</script>
</body>
</html>
```

Aber natürlich könnten solche eingebetteten Bereiche mit aktivem Code auch Funktionalitäten mit deutlich mehr Schadenspotenzial enthalten wie beispielsweise Funktionalitäten zum Auslesen von Cookies, Passwörtern oder weiteren Credentials. Denn je nach der konkreten Umsetzung eines Sandbox-Konzeptes des Browsers ist es mehr oder weniger schwer aus dieser Sandbox auszubrechen.

Ähnliche Risikobetrachtungen lassen sich für Web-Apps anstellen. Mehr und mehr Apps werden als Web-Apps realisiert. Sie sind eine jüngere Entwicklung von Apps und nutzen maßgeblich den Webbrowser zur Interaktion mit dem Benutzer. Dabei werden Web-Apps als Client-Server-Anwendung entwickelt. Ein wesentlicher Vorteil gegenüber herkömmlichen ‚nativen' Apps ist deren Plattformunabhängigkeit. Es müssen also nicht mehr wie bei der klassischen Entwicklung von Apps zwei oder gar mehrere Versionen für die mobilen Betriebssysteme Android, iOS oder weitere Zielplattformen von den Entwicklerteams implementiert werden. Anders als die herkömmlichen nativen Apps, die ja mehrheitlich in Java implementiert sind und ab und an nativen Code wie C oder C++ enthalten, werden Web-Apps mittels *Hypertext Markup Language Version* 5 (HTML5), JavaScript oder *Cascading Style Sheets* (CSS) implementiert. Einige Sicherheitsexperten bewerten eine derartige Entwicklung als positiv und argumentieren, dass damit durch zentrale Browser-Upgrades eventuelle Sicherheitslücken für alle Web-Apps gemeinsam beherrschbar werden. Negatives Testen mit einem sogenannten *Fuzzer* als automatisierte Technik für Softwaretests kann dazu dienen, Schwachstellen und Exploits in Browser-Engines zu finden und diese Zug um Zug durch Bereitstellen neuer Browser-Upgrades auszumerzen. Allerdings ignorieren sie dabei die Kehrseite der Medaille. Denn zum einen ist der Einsatz derartiger Fuzzer nicht allein das Privileg der ‚Guten' und wird gerne auch zur Entdeckung und Nutzung von Exploits von Angreifern

verwendet. Davon sind dann alle Web-Apps betroffen. Darüber hinaus bedeutet der Einsatz von Web-Apps auch, dass sich die Entwickler von Web-Apps nun auch mit den klassischen Angriffsformen von Webanwendungen auseinandersetzen müssen und sich dem dadurch teils massiven Schadenspotenzial stellen müssen, das von clientseitigen Angriffsformen wie *Reflected Cross Site Scripting* (XSS), *Persistent Cross Site Scripting, DOM-basierten Cross Site Scripting,* Click Jacking oder dem unsicheren Umbiegen von URLs ausgeht [4]. Erschwerend kommt hinzu, dass mit der im Jahre 2014 von der W3C verabschiedeten Version 5 von HTML die *Same-Origin-Policy (SOP)* aufgehoben wurde. Während die SOP besagt, dass es nur erlaubt ist, weitere Kommunikation zu derjenigen Domäne aufzubauen, von der auch die geladene Webseite stammt, gilt dies bei Cross-Origin Ressource Sharing nicht mehr. Es kann also ab HTML5 auch eine Verbindung zu einer anderen Domäne aufgebaut werden, die nicht der entspricht, von der die ursprüngliche geladene Webseite stammt. Dies macht es einem Angreifer deutlich einfacher, einmal erbeutete Credentials vom Opfersystem unerkannt abfließen zu lassen. Mit Unterstützung der SOP musste ein Angreifer diese zumindest überwinden, indem er für das unerkannte Abfließen-lassen von Credentials Techniken wie die Verwendung von HTML-Image-Tags und dergleichen einsetzten musste. Denn das Einbinden von Bildern hat von jeher die SOP unterlaufen mit der Argumentation, dass ansonsten Werbung von Drittanbietern auf einer Webseite nicht möglich wäre.

Spätestens mit dem Einzug von Web-Apps in das Ökosystem mobiler Geräte verschwimmen daher die Grenzen zwischen den Gebieten ‚Sicherheit für Webanwendungen' und ‚Mobile Security' und machen seriöse Risikobewertungen, denen mobile digitale Geräte und ihre Nutzer ausgesetzt sind, noch einmal erheblich unübersichtlicher. Wer einen Eindruck gewinnen möchte über die verschiedenen Angriffsarten und welche Best Practices und Maßnahmen zur Eindämmung ebendieser das Bundesamt für Sicherheit in der Informationstechnik empfiehlt, sei verwiesen auf [1] verwiesen. Spätestens dann dürfte offensichtlich sein, dass es mit dem einfachen Aufspielen von AdBlocker (Werbeblocker) für Smartphones bei weitem nicht getan ist.

11.2 Überwinden des Air-Gap

Im Vorwort dieses Buches haben wir einen Vergleich mit dem alpinen Wandern einer Seilschaft im Gebirge gezogen, um zu verdeutlichen, dass es eigentlich nur eine Möglichkeit gibt, sich in einer ‚vernetzten' Welt vor widrigen Impulsen und Angriffen von außen zu schützen: das *‚seilfreie Gehen'* oder in der IT-Welt ein mittels *‚air-gap'* abgeschottetes System. Gemeint ist bei Letzterem schlicht und einfach, der digitalen Außenwelt keine Kommunikationsschnittstelle anzubieten. Dass aber auch dies nicht ganz so einfach ist, wie man gemeinhin denken dürfte, wollen wir nun erörtern, indem wir eine App vorstellen, die der israelische Sicherheitsexperte Mordechai Guri und seine Kollegen entwickelt haben und im Jahre 2017 auf einer europäischen Konferenz für Cyber-Sicherheit in Norwegen einem breiteren Publikum vorgestellt haben [2]. Dabei

dient die Überwindung des Air-Gap einem Angreifer diesmal nicht dazu, den Nutzer des Smartphones auszuspionieren. Vielmehr ist es umgekehrt. Der Angreifer verwendet ein Smartphone, auf dem eine bestimmte App läuft, um die Luftstrecke (engl. *air-gap*) zu einem Rechner zu überwinden.

Dass in der Literatur bereits seit Dekaden eine ganze Reihe von Möglichkeiten eines Datenaustritts über die Luftstrecke in Abwesenheit von jeglicher drahtgebundener oder drahtlosen Netzwerktechnologie diskutiert werden, offenbart, dass ein mittels *‚air-gap‘* gesichertes System keinesfalls mit seiner hermetischen Abriegelung gleichzusetzen ist. Elektromagnetische Abstrahlung von Komponenten, aber auch optische und thermische Eigenschaften bieten oftmals die Möglichkeit, einen Seitenkanal aufzubauen, über den, wenn auch nur mit geringen Reichweiten und sehr kleinen Datenraten, Informationen abfließen können. In [2] haben die Sicherheitsexperten aufgezeigt, wie man unter Verwendung von akustischen Eigenschaften des Opfergerätes Daten abfließen lassen kann, und zwar auch dann noch, wenn die Sicherheitsbestimmungen eines Unternehmens die Verwendung von Lautsprechern und Mikrofonen an deren Rechnern strikt untersagen.

Der Ansatz *DiskFiltration* macht sich die Tatsache zunutze, dass das Festplattenlaufwerk (engl. *Hard Disk Drive* (HDD)) eines Rechners Geräusche erzeugt. Dabei ist das Festplattenlaufwerk das am meisten verwendete Speichermedium für PCs, Server und auch Laptops. Die Entwickler von DiskFiltration weisen in ihrer Ausarbeitung jedoch explizit darauf hin, dass ihr Ansatz nicht funktioniert, wenn ein *Solid State Drive* (SSD) oder ein *Solid State Hard Drive* (SSHD) zum Einsatz kommen würde. Bei einem HDD aber erzeugt der mechanische Arm eines Festplattenlaufwerks bei seiner Bewegung zur Durchführung von Lese- und Schreiboperationen typische Geräusche, um die korrekte Position auf der magnetischen Platte zu finden. Dies ist auch dann noch der Fall, wenn spezielle Verfahren zur Eindämmung derartiger Geräusche des Festplattenlaufwerks aktiviert sind. Konkret ist der Angriff daher auch bei einer Aktivierung von *Automatic Acoustic Management* (AAM) möglich.

Für den Angriff auf einen gewöhnlichen Computer mit HDD – jedoch ohne installierten Lautsprecher und Mikrofon – ist eine auf dem Computer eingeschleuste Malware erforderlich, welche die Daten zusammenstellt, die das abgeschottete System verlassen sollen. Da ein mit DiskFiltration erzeugter Seitenkanal allerdings bestenfalls geeignet ist, kleinere Datenmengen zu übertragen, dürfte es sich hierbei typischerweise um ein Passwort oder aber Schlüsselmaterial handeln. Das empfangende Gerät muss über die Fähigkeit verfügen, Geräusche aufzunehmen. Es kann also ein herkömmliches Smartphone, eine Smartwatch oder dergleichen sein, die möglichst nicht weiter als in einem Abstand von einem bis maximal zwei Meter neben dem (Opfer)-System abgelegt wird.

Bei den vom HDD erzeugten Geräuschen ist zwischen zwei wesentlichen Geräuschquellen zu unterscheiden: Zum einen ist dies der Motor, zum anderen sind dies Geräusche, die von der Bewegung des Armes herrühren. Glücklicherweise sind die Geräusche, die beim Zugriff (engl. *seek*) durch die Bewegung des mechanischen Armes erzeugt werden, lauter als die Geräusche des Motors im Leerlauf (engl. *idle*). Lese- und Schreiboperationen erhöhen die Frequenz für eine kurze Zeit. Durch eine anschließende

Analyse einer so erzeugten Wellenform, beispielsweise über eine Fast Fourier-Transformation (FFT), kann dann die Frequenz der Geräusche auf der Empfängerseite analysiert werden. Nun wird auch klar, warum der Abstand zum Opfersystem so gering als möglich sein sollte: Ansonsten würde das Hintergrundrauschen dominieren und die Geräusche der Armbewegung zu stark überlagern, als dass sie noch empfängerseitig mit Komponenten aus dem Consumer-Bereich analysierbar wären.

In [2] haben die Sicherheitsexperten für die Schadsoftware auf dem Opfersystem und die App auf dem Smartphone des Angreifers die folgende Umsetzung vorgestellt:

1. Auf dem Opfergerät:
   Die Schadsoftware kann als Nutzerprozess ohne weiterführende Rechte gestartet werden. Denn zur Umsetzung der Zugriffsoperationen seek() und einer damit einhergehenden Bewegung des mechanischen Armes zur Erzeugung akustischer Signale kann beispielsweise ein C-Programm unter Verwendung von Dateiaufrufen wie fopen() verwendet werden. Für die Umsetzung wäre auch ein Shell-Skript denkbar welches den Befehl dd zum Kopieren von Daten anwendet. Der Kern des Ansatzes besteht dann in der Umsetzung einer sogenannten *On-Off-Keying-(OOK-)*Modulation, mit der binäre Daten mithilfe besagter HDD-Geräusche zu übertragen sind. Dabei wird ein zu übertragendes 0er-Bit als eine Zeitspanne T_0 dargestellt und ein 1er-Bit als die Zeitspanne T_1. In der Umsetzung erfolgt dann für ein 0er Bit ein sleep() für die Zeitspanne T_0, in der der mechanische Arm nicht bewegt wird und seine Geräusche somit die des Motors nicht überlagern werden. Soll hingegen ein 1er-Bit übertragen werden, dann wird der mechanische Arm in Bewegung gebracht, indem innerhalb der Zeitspanne T_1 wieder und wieder möglichst entfernte Positionen innerhalb des Sektors einer Platte angesteuert werden. In Abb. 11.1 ist dies über die Parameter *Sektorbeginn* und *Sektorende* dargestellt. Um anzudeuten, dass die Übertragung in diesem Fall über einen Seitenkanal erfolgt, wurde in der Darstellung das Symbol ~ verwendet, zur Unterscheidung zu der in den Grundlagen eingeführten Notation → für eine intendierte drahtlose Übertragung (siehe Teil 2).
   Darüber hinaus schlagen die Sicherheitsexperten vor, dass die Malware auf dem Opfergerät vorab eine Art Mini-Rahmen sendet, der neben den konkreten Werten für die Zeitspannen T_0 und T_1 auch zur Synchronisation und zum Abtasten des Frequenzbereiches zwischen Malware und Spionage-App auf dem Smartphone dient. Nur so ist es der App möglich den Träger für die OOK-Modulation zu erwischen. Dieser Aspekt ist in Abb. 11.1 jedoch nicht aufgeführt.
2. Spionage-App auf dem Smartphone:
   Eine App muss nun in der Lage sein, das Audiosampling durchzuführen, den Frequenzbereich abzutasten, um danach beispielsweise unter Anwendung der Fast Fourier-Transformation (FFT) die Frequenz der Geräusche auf der Empfängerseite zu analysieren. Für die Implementierung dieser OOK-Demodulation kann beispielsweise der Spatula-Quellcode [5] verwendet und angepasst werden. Der Aufruf von $OOKDemodulation(T_0, T_1)$ unter Verwendung der Zeitspannen T_0 und T_1 deutet darauf

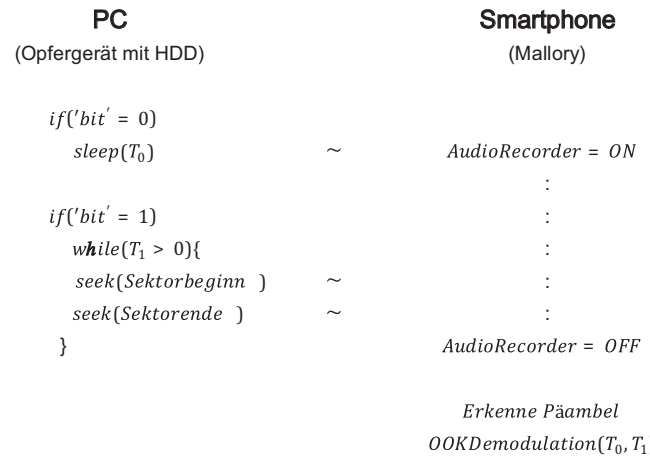

Abb. 11.1 Überwindung des Air-Gap mittels Malware und Spionage-App DiskFiltration. (Quelle: eigene Darstellung in Anlehnung an [2])

hin, dass zu diesem Zeitpunkt mit dem Erkennen der Präambel im Mini-Rahmen Mallory auch die für die Demodulation notwendigen Zeitspannen für 0er- und 1er-Bits vorliegen.
Neben der Implementierung der israelischen Sicherheitsforscher, die diesen Ansatz vorgestellt haben, hat der Student Fabian Holzer im Rahmen seiner Bachelorarbeit an der Hochschule Offenburg ebenfalls eine derartige App erstellt und evaluiert [3].

Da das geschilderte Verfahren DiskFiltration einem Angreifer Mallory tatsächlich nur ermöglicht, über eine kurze Distanz von 1–2 m nur sehr wenige Daten pro Zeiteinheit zu erhalten – die Sicherheitsexperten haben in ihrer PoC-Implementierung $T_0 = 2s$ und $T_1 = 1s$ gewählt – mag je nach Paranoid-Level den einen Leser beruhigen, wieder andere hingegen eher dennoch bedenklich stimmen. Auch kann dieser Abschnitt nur als ein erster Denkanstoß für die Leserschaft zum Thema ‚Überwindung des Air-Gap' verstanden werden. So dürften sicherlich deutlich leistungsstärkere Ansätze für Seitenkanäle mit höheren Datenraten und weiteren Distanzen verfügbar sein.

11.3 Zusammenfassung

Im Rahmen dieses Kapitels haben wir uns zunächst mit Technologien wie Web-Apps und QR-Codes beschäftigt. Hierüber werden Verbindung zu entfernten Servern erzeugt und der Nutzer von Web-Apps oder auch QR-Codes könnte dadurch Opfer eines Drive-by-download-Angriffs werden.

Spätestens mit dem Einzug von Web-Apps in das Ökosystem mobiler Geräte verschwimmen somit die Grenzen zwischen Sicherheitsfragen für Webanwendungen und mobiler digitaler Geräte. Dies macht eine seriöse Risikobewertungen, denen mobile digitale Consumer-Geräte und ihre Nutzer ausgesetzt sind, noch einmal anspruchsvoller.

Innerhalb des zweiten Teils des Kapitels haben wir erörtert wie findige Sicherheitsexperten mittels einer App auf dem Smartphone und einer Malware auf einem PC versuchen den air-gap zwischen diesen Geräten zu überwinden. Der Ansatz DiskFiltration macht sich die Tatsache zunutze, dass das Festplattenlaufwerk eines Rechners Geräusche erzeugt.

Durch kontrollierte Bewegungen des mechanischen Plattenarmes erzeugt eine Malware auf dem PC Geräusche welche mittels einer App auf dem Smartphone als Bitfolge interpretiert werden können. Die Sicherheitsexperten haben gezeigt wie auf diese Weise über kleine Distanzen geringe Datenmengen unbemerkt von dem Opfergerät abfließen können.

Literatur

1. BSI: Sicherheit von Webanwendungen – Maßnahmenkatalog und Best Practices. https://www.bsi.bund.de/DE/Publikationen/Studien/Websec/index_htm.html. Zugegriffen: 28. Nov. 2019
2. Guri, M., Solewicz, Y., Daidakulov, A., Elovici, Y.: Acoustic, data exfiltration from speakerless air-gapped computers via covert hard-drive noise. ESORICS, Norway (2017)
3. Holzer, F.: Acoustic data exfiltration via hard drive noise. Kolloquium Werkschau M+I. https://werkschau.mi.hs-offenburg.de/kolloquium/. Zugegriffen: 26. Nov. 2019
4. OWASP: Web application security guidance. https://www.owasp.org/index.php/Web_Application_Security_Guidance. Zugegriffen: 28. Nov. 2019
5. Spatula: Modulating and demodulating signals in Java. http://spatula.net/mt/blog/2011/02/modulating-and-demodulating-signals-in-java.html. Zugegriffen: 22. Nov. 2019

Positionsbestimmung und Standortverfolgung

12

Zu Beginn des vierten Teils dieses Buches haben wir bereits darauf hingewiesen, dass wir uns zum Ende dieses Teils noch einmal mit weiteren Fähigkeiten eines modernen mobilen Betriebssystems auseinandersetzen wollen. Wir haben im vorherigen Kapitel erörtert, welche Möglichkeiten ein App-Entwickler besitzt, seine App weitgehend unerkannt mit Hintertüren (engl. *backdoor*) auszustatten, und dass die Analyse solcher Apps auch für erfahrene Sicherheitsexperten und *Reverse Engineers* keinesfalls trivial ist. Dies ist sicherlich bedenklich und unerfreulich. Jedoch hätte ein paranoider Smartphone-Nutzer hier zumindest die Möglichkeit, keinerlei Apps zu installieren und zu starten. Was aber, wenn das mobile Betriebssystem selbst über Funktionalitäten verfügt, die wohl zumindest für die meisten seiner Nutzer weitgehend verborgen und unbekannt sind? Uns interessieren insbesondere solche Aktivitäten, die moderne mobile Betriebssysteme wie Android, aber auch iOS permanent sozusagen hinter unserem Rücken durchführen. Dabei gehen wir davon aus, dass unser Smartphone-Nutzer noch ein wenig paranoider ist. Er hat nicht nur wie oben bereits erwähnt keinerlei Apps installiert, sondern er achtet auch tunlichst darauf, weder WLAN, Bluetooth, NFC noch jedwede weitere drahtlose Kommunikationsart aktiviert zu haben, da er sonst befürchten würde, *jemand* könne seine aktuelle Position in Erfahrung bringen. Natürlich hat unseren Nutzer auch der in Abschn. 4.2 dargestellte Privacy-Schutz bei Bluetooth nicht überzeugt. Denn er weiß, dass sich hierbei Bewegungsprofile auch dann noch nachvollziehen lassen, wenn die MAC-Adresse entweder durch zufällig gewählte Adressen oder eine häufige Abänderung verschleiert ist.

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_12

12.1 Verfahren zur Positionsbestimmung und Standortverfolgung

Durch die Nutzung eines mobilen digitalen Gerätes mit verschiedensten verbauten drahtlosen Übertragungstechnologien sind unsere Nutzerstandorte also näherungsweise nachvollziehbar. Dies kann auf unterschiedliche Art und Weise geschehen, wobei nicht immer die gleiche Partei von der Standortberechnung profitiert. Dabei werden die Techniken durchgeführt, unabhängig davon, ob der Nutzer der Verwendung von *Location Based Service* (LBS) zugestimmt hat oder nicht.

Bevor wir insbesondere auf zwei Techniken zur Positionsbestimmung eingehen, hier noch eine Anmerkung zur Widersprüchlichkeit des Nutzerverhaltens: Laut [1] nutzen 67 % der Smartphone-Nutzer Apps, indem sie aktiv den Zugriff auf ihre Standortdaten gewähren. Allerdings geben auch 64 % der Nutzer an, es nicht zu mögen, derart transparent zu sein.

Bitte beachten Sie: Die nun geschilderten Technologien funktionieren unabhängig davon, ob der Nutzer eines Smartphones den verwendeten Apps Zugriff auf seine Standortpositionen gewährt hat oder nicht. Darüber hinaus werden wir sehen, dass es bei genauerem Hinschauen bei einem dieser Ansätze nicht nur um das Nachvollziehen von aktuellen Benutzerstandorten geht, sondern hier unter Umständen ein wenig globaler gedacht werden sollte. Doch mehr zu diesem Thema dann in Abschn. 12.1.3.

12.1.1 Positionsbestimmung und Standortverfolgung mittels aktiven WLAN-Scanning

Nationale Telekommunikationsanbieter haben im Falle eines Verbindungsaufbaus die Verbindungsdaten zu speichern, so, dass sie erkennen, über welche Radiozelle(n) der Nutzer des mobilen Gerätes seine Kommunikation tätigte. Damit erhalten Telekommunikationsanbieter ein mehr oder weniger akkurates Bewegungsprofil, solange sich der Nutzer eines Gerätes mit dem Gerät im Abdeckungsbereich des Anbieters aufhält. Aufgrund relativ großer Zellradien wie sie insbesondere im ländlichen Raum vorzufinden sind, ist hierüber jedoch nur eine verhältnismäßig grobgranulare Standortverfolgung (engl. *tracking*) eines mobilen Nutzers möglich.

Darüber hinaus agieren viele Smartphones oder, allgemeiner ausgedrückt, mobile digitale Geräte mit WLAN-Interface zum Auffinden eines WLAN-Access-Points im *aktiven Modus.* Dabei ist der aktive Modus die Voreinstellung. Dies ist bedauerlich, da unter Verwendung des *passiven Modus* das anonyme Auffinden eines WLAN-Access-Points prinzipiell möglich wäre. Denn mit diesem Verfahren würden bei der Suche nach einem WLAN-Access-Point selber keinerlei Nachrichten auf der sogenannten Protokollebene 2, die L2-Beacons, vom mobilen Gerät versendet werden. Für Geräte im aktiven WLAN-Scanning-Modus dagegen werden sehr wohl derartige Nachrichten versendet. Deren WiFi-Karte versendet alle paar Millisekunden sogenannte *Broadcast-Probe-Anfragen,* um einen

WLAN-Zugangspunkt in der Nähe ausfindig zu machen. Da solch eine Beacon-Anfrage auf der Sicherungsschicht neben dem *Basic Service Set Identifier* (BSSID) und dem *Service Set Identifier* (SSID) auch die als weltweit eindeutig zu betrachtende MAC-Adresse enthält, wird hierüber ebenfalls eine Positionsbestimmung des mobilen Gerätes und seines Trägers möglich. In Abb. 12.1 veranschaulichen wir den *passiven Scanning-Modus* unter Sendung der Kontrollrahmen der Nachrichtentypen ‚probe request', ‚association request' sowie einen ‚association response'. Der Lesbarkeit halber sind in den Nachrichten *ProbReq*(), *AssReq*() und *AssRes*() nur jeweils die wesentlichen Parameter berücksichtigt wobei wir uns bei diesen an die in Abschn. 2.1 eingeführte Schreibweise gehalten haben. Die Reduzierung auf die wesentlichen Parameter dürfte für das prinzipielle Verständnis jedoch hoffentlich nicht von Nachteil sein. Neben einer Basisstation BS_A ist noch eine weitere Basisstation BS_B in der Abbildung dargestellt. Die Basisstation BS_B befindet sich dabei ebenfalls in der Nähe des Smartphones *SP*. Allerdings liegt diese nicht mehr in der gegenseitigen Übertragungsreichweite. Das Zeichen !← deutet dabei an, dass diese Basisstation zwar eine Nachricht ausgesandt hat, diese jedoch nicht vom Smartphone empfangen wurde. Die Abb. 12.2 zeigt hingegen die Abfolge der Kontrollnachrichten auf der Sicherungsschicht für den *aktiven Scanning-Modus,* der, wie gesagt, in der Voreinstellung vom mobilen Gerät angestoßen wird und sich daher zur Positionsbestimmung und zum Verfolgen des Gerätes bzw. des Nutzers des mobilen Gerätes eignet.

Dadurch, dass die meisten mobilen Geräte zum Auffinden eines WLAN-Access-Points das WLAN-Scanning im aktiven Modus durchführen, wird die Position des mobilen Gerätes für den Betreiber des Access-Points sichtbar. Solch eine Positionsbestimmung ist aufgrund der deutlich kleineren WLAN-Übertragungsradien gegenüber

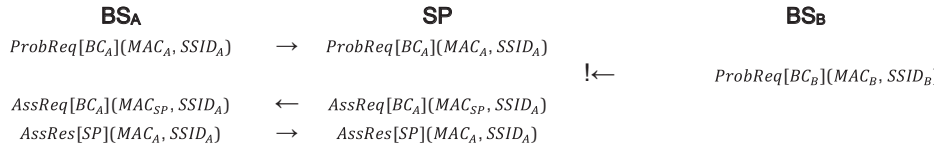

Abb. 12.1 WLAN-Scanning im *passiven* Modus. (Quelle: eigene Darstellung)

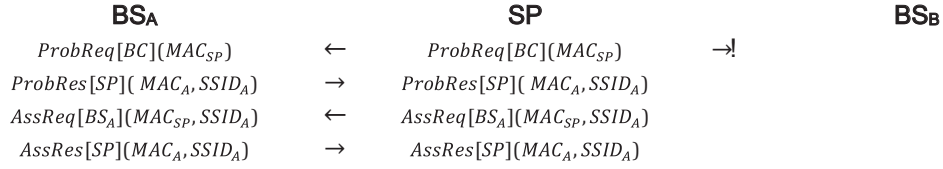

Abb. 12.2 WLAN-Scanning im *aktiven* Modus. (Quelle: eigene Darstellung)

denen zellularer Mobiltelefonie deutlich akkurater. Damit ist das WLAN-Tracking eines Smartphone-Nutzers immer dann möglich, wenn eine Menge von WLAN-Zugangspunkten demselben Hoheitsgebiet angehören und MAC-Adressen sowie Zeitstempel gespeichert werden. Dies ist an Flughäfen, in Einkaufshäusern, Universitäten und anderen Plätzen der Fall bzw. dort wo verschiedene dieser Hoheitsgebiete miteinander kooperieren und die Daten korreliert werden können. Dabei ist es sicherlich auch von Interesse, dass ein derartiges Verfolgen eines Smartphone-Nutzers durch Broadcast-Probeanfragen auch dann noch möglich ist, wenn das Gerät sich gar nicht bei einem WLAN-Zugangspunkt eingewählt hat. Denn der Vorgang des WLAN-Scanning erfolgt unabhängig von der Einwahl bzw. dem eigentlichen Übertragen von Nutzdaten, wie etwa Webanfragen des Nutzers oder dergleichen. Da der hier geschilderte Ansatz auch innerhalb von Gebäuden funktioniert und sehr leicht umsetzbar ist, ist dies darüber hinaus ein klarer Vorteil gegenüber beispielsweise rein satellitengestützten Technologien wie dem *Global Positioning System* (GPS). Denn diese benötigen mindestens vier Satelliten in ungestörter Sichtverbindung zum Empfänger.

Aus den oben geschilderten Gründen wurde – vermutlich auch getrieben durch die Standardisierungsgremien der Telekommunikationsanbieter wie die *Open Mobile Alliance* (OMA) und das *3rd Generation Partnership Project* (3GPP) – das Verfahren *Assisted-GPS* standardisiert. Damit können neben den GPS-Informationen auch weitere Informationen von Sensoren am Smartphone in seine Positionsberechnung einfließen, wie zum Beispiel die des magnetischen Kompasses, von Sensoren zur Beschleunigungsmessung, aber auch Informationen weiterer Schnittstellen zur Mobilkommunikation wie WLAN, Bluetooth oder NFC. Dass hierfür jedoch die Einbindung der Telekommunikationsanbieter selbst gar nicht mehr erforderlich ist, zeigt der nachfolgend beschriebene Weg den die Unternehmen Google und Apple mit ihren mobilen Betriebssystemen Android und iOS beschritten haben.

▶ Das WLAN-Scanning im aktiven Modus als die Voreinstellung bei der Suche nach einem WLAN-Access-Point ermöglicht dem Betreiber des WLAN eine recht genaue Approximation der aktuellen Position eines Gerätes auf der Basis von in den einzelnen Beacon-Nachrichten enthaltenen MAC-Adressen des Smartphones oder anderen WLAN-fähigen Geräten. Beachtenswert ist, dass eine derartige **Positionsbestimmung auch ohne tatsächlich erfolgten Verbindungsaufbau möglich** ist.

12.1.2 Positionsbestimmung und Standortverfolgung mittels Swarm-Mapping

Andreas Dhein und Rüdiger Grimm [2] haben im Jahre 2017 aufgezeigt, wie einige mobile Betriebssysteme eines Smartphones durch die Auswertung der *Received Signal Strength Indication (RSSI)* ihre Distanz zu einer Basisstation oder aber zu einem

WLAN-Access Point durch die Verwendung eines sogenannten Swarm-Mapping-Algorithmus approximieren. Eigentlich dient ein RSSI-Wert zur Abschätzung der Empfangsfeldstärke drahtloser Kommunikation. Er wird vornehmlich vom mobilen Gerät zur Auswahl eines brauchbaren Kanals genutzt. Für Radiozellen liegen typische RSSI-Werte im Bereich 25–50 dBm, für WLAN im Bereich 20–90 dBm.

Neben der Auswahl des Kanals kann der RSSI-Wert aber auch in Verfahren zur Positionsbestimmung eingehen. Die mobilen Betriebssysteme iOS und Android verwenden diesen Wert tatsächlich auch hierfür. Die genaue Berechnung ist allerdings sehr komplex und darüber hinaus vielfältigen Störungen ausgesetzt. Daher kann sie in erster Instanz nur näherungsweise erfolgen. So kann sich die Topografie der Umgebung oder aber einfach die Handstellung des Smartphone-Nutzers signifikant auf die Genauigkeit einer so durchgeführten Approximation auswirken. Die genauen Algorithmen werden vom Hersteller geheim gehalten. Offenkundig ist hingegen die prinzipielle Vorgehensweise, die Andreas Dhein und Rüdiger Grimm in [2] herausgestellt haben. Dabei unterscheiden sie zwischen einer intrinsischen und einer extrinsischen Lokalisierung:

Intrinsische Lokalisierung Aktuelle Smartphones sammeln Informationen über Sendestationen (WLAN-Access-Point, Radio-Basisstation) in ihrer unmittelbaren Nähe und gleichen diese mit ihren aktuellen über GPS erfassten Koordinaten ab. Die so erhaltenen Informationen sendet das Smartphone von Zeit zu Zeit in verschlüsselter Form an das jeweilige Unternehmen Google bzw. Apple. Dort werden diese Informationen in einer Datenbank abgelegt.

Extrinsische Lokalisierung Die mittels der intrinsischen Lokalisierung von vielen Smartphones erhaltenen und gesammelten Positionsinformationen werden wiederum an andere Smartphones übermittelt. Dies dient dem vordergründigen Ziel, deren eigene Position akkurater zu berechnen.

Ein von Andreas Dhein [2] entwickeltes Analysewerkzeug für das mobile Betriebssystem iOS ergibt nun, dass das Smartphone zuerst den Übertragungsradius der aktuell verbundenen Basisstation anfragt und lokal eine Datenbank über die Zeit mit Informationen zu seiner eigenen approximierten Position aufbaut. Diese Datenbank enthält Einträge zu den folgenden Kategorien:

- Zeittempel
- Breitengrad
- Längengrad
- Horizontale Genauigkeit
- Höhengrad
- Vertikale Genauigkeit
- Geschwindigkeit
- Richtung
- TripID

In einem zweiten Schritt erhält das Smartphone dann vom Apple-eigenen Cloud-Dienst die Positionen anderer Basisstationen und Access-Points in der Umgebung des anfragenden Smartphones. Dies dient einer ersten grobgranularen Lokalisierung, bevor in einem weiteren Schritt dann die Positionsapproximation auf Basis von Positionen der WLAN-Access-Points und Basisstationen verfeinert werden kann.

Auch wenn, wie oben bereits erwähnt, die einzelnen verwendeten Algorithmen nicht offengelegt sind, so wird das genutzte Messverfahren zur Positionsbestimmung ein Trilaterationsverfahren sein. Anders als bei Triangulationsverfahren, die die Positionsbestimmungen auf der Grundlage dreier Winkel durchführen, wird bei den Trilaterationsverfahren die jeweiligen Abstände dreier Punkte zueinander für die Positionsbestimmung verwendet.

So *könnte* das Verfahren auch den Satz von Pythagoras in die Berechnung eingehen lassen. Wir erinnern uns: Laut dem Satz von Pythagoras ist bei einem rechtwinkligen Dreieck die Summe der quadratischen Flächen über den Katheten mit Längen a und b gleich der quadratischen Fläche über der Hypotenuse, also $a^2 + b^2 = c^2$ oder, ein wenig anders ausgedrückt, $c = \sqrt{a^2 + b^2}$. Bezeichnen wir nun für den zweidimensionalen Fall die Position der Basisstation *BS* mit (x_{BS}, y_{BS}) über ihre x-Koordinate und ihre y-Koordinate und die aktuelle Position des Smartphones *SP* mit (x_{SP}, y_{SP}) auf die gleiche Weise, so lassen sich die folgenden Beziehungen für die Längen der Katheten des rechtwinkligen Dreiecks aufführen als $a = |x_{SP} - x_{BS}|$ und $b = |y_{SP} - y_{BS}|$. In dieser zugegeben etwas akademischen Darstellung entsprechen die x-Koordinaten und die y-Koordinaten also den Datenbankeinträgen der jeweiligen Längen- und Breitengrade (die dritte Dimension, der Höhengrad ist der Einfachheit halber hierbei nicht in die Betrachtung mit einbezogen). Die Länge der Hypotenuse c kann man nun darstellen, indem man den RSSI-Wert verwendet. Da über den RSSI-Wert jedoch die Distanz zwischen der Basisstation und dem Smartphone nur approximiert werden kann notieren wir diesen Sachverhalt mit $c \approx d(RSSI)$. Insgesamt ergibt sich demnach die folgende Beziehung

$$d(RSSI) \approx \sqrt{|x_{SP} - x_{BS}|^2 + |y_{SP} - y_{BS}|^2}$$

Dieser Sachverhalt ist in Abb. 12.3 noch einmal aufbereitet:

Bitte beachten Sie, dass das soeben geschilderte Verfahren der Trilateration basierend auf dem Satz von Pythagoras, zwei Ebenen einer Korrelation bereithält:

1. Verschiedene derart approximierte Distanzen $c = d(RSSI)$ eines einzelnen Smartphones *SP* werden zu unterschiedlichen Zeitpunkten t_i korreliert (intrinsische Lokalisierung).
2. Verschiedene approximierte Distanzen $c = d(RSSI)$ vieler Smartphones SP_i werden zueinander in Bezug gesetzt (extrinsische Lokalisierung).

Diese Korrelation vieler RSSI-Werte führt dazu, dass auf der Grundlage solch eines in [2] genannten Swarm-Mapping-Ansatzes trotz vielfältiger Störungen eine recht genaue Positionsbestimmung ermöglicht wird.

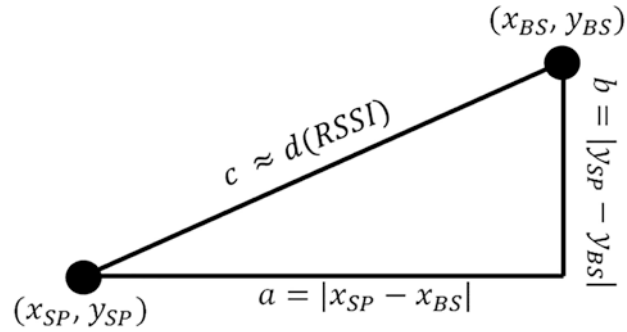

Abb. 12.3 Grafische Darstellung des Ausdrucks $d(RSSI) \approx \sqrt{|x_{SP} - x_{BS}|^2 + |y_{SP} - y_{BS}|^2}$ aus dem Satz von Pythagoras. (Quelle: Eigene Darstellung)

▶ **Wichtig**
warm-Mapping ermöglicht eine recht genaue Approximation der aktuellen Position eines Gerätes auf der Basis von am Smartphone gemessenen RSSI-Werten einer Basisstation oder eines WLAN-Access-Points zur Distanz $d(RSSI)$ zwischen Smartphone SP und Basisstation BS. Mithilfe wiederholter Trilateration über die Zeit durch Anwendung des Satzes von Pythagoras

$$d(RSSI_i) \approx \sqrt{|x_{SP(ti)} - x_{BS}|^2 + |y_{SP(ti)} - y_{BS}|^2}$$

erfolgt eine Positionsbestimmung des Smartphones, die kontinuierlich den Herstellern der mobilen Betriebssysteme mitgeteilt wird. Hierbei beschreiben $(x_{SP(ti)}, y_{SP(ti)})$ die Position des Smartphones zum Zeitpunkt t_i und (x_{BS}, y_{BS}) die Position der Basisstation. Beachtenswert ist, dass eine derartige **Positionsbestimmung auch ohne Verbindungsaufbau möglich** ist.

12.1.3 Positionsbestimmung und Standortverfolgung durch Swarm-Mapping+

Allerdings hat man es mit dem oben beschriebenen Verfahren der Trilateration mit einem klassischen Henne Ei Problem zu tun: i) Hat man die Position des Smartphones bestimmt, so lässt sich die Position der Basisstation bzw. je nach drahtloser Übertragungstechnologie die Position des WLAN-Acssess-Points berechnen, und ii) kennt man die Position von Basisstation oder des WLAN-Acssess-Points, so lässt sich die Position des Smartphones berechnen.

Diese Henne-Ei-Problematik entschärft sich jedoch deutlich bei genauerem Hinsehen aufgrund der folgenden Gegebenheiten:

Ad1 Die Position der Basisstation ist statisch und typischerweise unveränderbar.

Ad2 Jedes Smartphone misst viele RSSI-Werte zu unterschiedlichen Zeitpunkten und diese von oftmals unterschiedlichen Positionen.

Ad3 Jedes Smartphone verfügt noch über eine Reihe von weiteren Sensoren, die ebenfalls in die Bestimmung des Standortes eingehen können.

Ad4 Tausende weiterer Smartphones mit einem mobilen Betriebssystem des gleichen Herstellers helfen ebenfalls bei einer derartigen Approximierung.

Wir wollen uns diese zwei Ebenen der Korrelation anhand eines sehr einfachen Beispiels mit zwei Smartphones SP_1 und SP_2 vergegenwärtigen, aber nun unter der Annahme, dass dem Smartphone zum Zeitpunkt t der Messung von RSSI-Werten seine eigenen, der Einfachheit halber zweidimensionalen Positionsdaten $(x_{SP(t)}, y_{SP(t)})$ bereits näherungsweise vorliegen. Neben dem eigentlichen Swarm-Mapping-Verfahren auf Basis von Trilateration gehen in die Bestimmung der Positionsdaten des Smartphones eben auch GPS-Informationen und Messwerte der im Smartphone verbauten Sensoren ein.

Die Bewegungsmuster beider Geräte und deren Abtastzeitpunkte für die verschiedenen RSSI-Werte seien dabei gegeben, so wie sie in Abb. 12.4 dargestellt sind. Das Smartphone SP_1 nimmt zu drei unterschiedlichen Zeitpunkten zur Abschätzung der Empfangsfeldstärke RSSI-Werte entgegen, während es sich in der Nähe einer Basisstation BS aufhält. Da es sich zum Zeitpunkt t_3 bereits außerhalb des Übertragungsradius von BS befindet, kann es nur die Werte $RSSI_1$ und $RSSI_2$ zu den Messzeitpunkten t_1 und t_2 in die Berechnung eingehen lassen. Ähnliches lässt sich für das zweite Smartphone SP_2 in der Nähe der Basisstation feststellen. Dieses befindet sich nur zu den Messzeitpunkten t_5 und t_6 innerhalb des Übertragungsradius, sodass die $RSSI$-Werte zu den Zeitpunkten t_4 und t_7 für diese Basisstation nicht verfügbar sind. Damit ergeben sich für die beiden Smartphones die folgenden Gleichungen, die in deren jeweilige Trilaterationsverfahren eingehen.

Das mobile Betriebssystem von Smartphone SP_1 kann eigenständig und lokal auf dem Gerät folgende Werte berechnen:

$$d(RSSI_1) \approx \sqrt{|x_{SP(t1)} - x_{BS}|^2 + |y_{SP(t1)} - y_{BS}|^2}$$

$$d(RSSI_2) \approx \sqrt{|x_{SP(t2)} - x_{BS}|^2 + |y_{SP(t2)} - y_{BS}|^2}$$

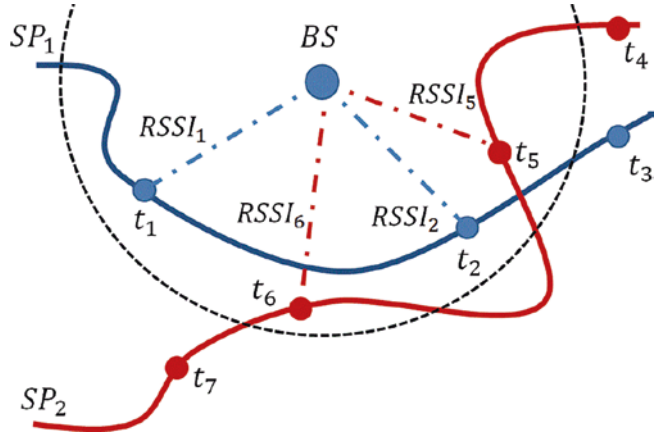

Abb. 12.4 Beispielhafte *RSSI*-Werte zweier Smartphones zu verschiedenen Zeitpunkten t. (Quelle: eigene Darstellung in Anlehnung an [3])

Das mobile Betriebssystem von Smartphone SP_2 kann seinerseits eigenständig und lokal auf dem Gerät diese Werte berechnen:

$$d(RSSI_5) \approx \sqrt{|x_{SP(t5)} - x_{BS}|^2 + |y_{SP(t5)} - y_{BS}|^2}$$

$$d(RSSI_6) \approx \sqrt{|x_{SP(t6)} - x_{BS}|^2 + |y_{SP(t6)} - y_{BS}|^2}$$

Eine grafische Darstellung der für SP_1 verfügbaren Informationen aus der ersten Korrelationsstufe veranschaulicht recht prägnant, dass hier oftmals eine ungenügende Datenbasis für ein einzelnes Smartphone vorliegt, um eine eindeutige Aussage zur Position der Basisstation treffen zu können. Im vorliegenden Beispiel sind es zwei Positionen, an denen der Access-Point oder die Basisstation positioniert sein könnte. Eine ähnliche Schlussfolgerung kann das zweite Smartphone mittels des ihm lokal vorliegenden Datenbestandes ziehen. Auch hier ergeben sich unter Verwendung von $RSSI_5$ und $RSSI_6$ zwei mögliche Positionen, die sich aus den Schnittpunkten, oder besser Bereichen, der approximierten Distanzen $d(RSSI_5)$ und $d(RSSI_6)$ ergeben. Die Ergebnisse der ersten Korrelationsstufen an den Smartphones SP_1 und SP_2 sind in den Abb. 12.5 und 12.6 noch einmal illustriert. Jedes Smartphone erhält aufgrund seiner beschränkten lokalen Datenbasis als Ergebnisse zwei mögliche Standorte für die Basisstation.

Allerdings weisen Dhein und Grimm in ihren Untersuchungen zu den mobilen Betriebssystemen Android und iOS darauf hin, dass sich nach dieser ersten Korrelationsstufe, die sie intrinsische Lokalisierung genannt haben, noch eine zweite Stufe der

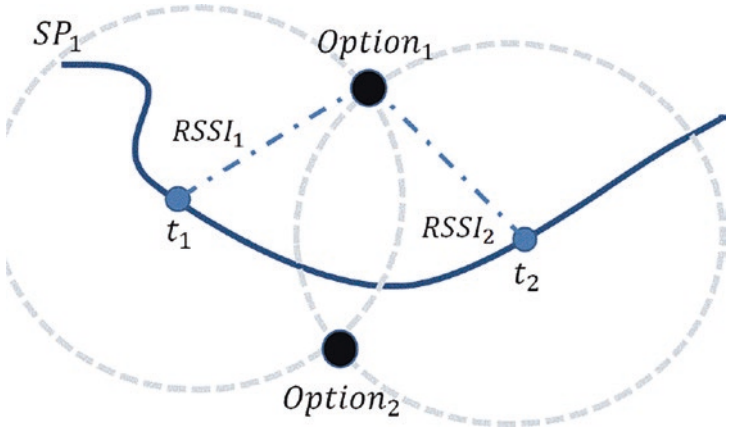

Abb. 12.5 Ergebnis der ersten Korrelationsstufe: SP_1 erhält zwei Optionen für eine mögliche Position der Basisstation. (Quelle: eigene Darstellung in Anlehnung an [3])

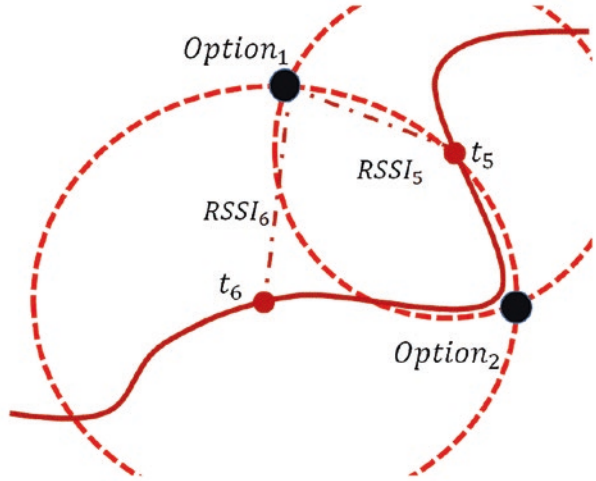

Abb. 12.6 Ergebnis der ersten Korrelationsstufe: Auch SP_2 erhält zwei Optionen für eine mögliche Position der Basisstation. (Quelle: eigene Darstellung in Anlehnung an [3])

Korrelation anschließt, die extrinsische Lokalisierung. Mit dieser zweiten Stufe der Korrelation, bei der verschiedene approximierte Distanzen $d(RSSI)$ vieler Smartphones zueinander in Bezug gesetzt werden, ergibt sich für die Anbieter Google und Apple von mobilen Betriebssystemen dann eine eindeutige Position der Platzierung der Basisstation. Auch dieser Sachverhalt sei der Vollständigkeit halber für unser Beispiel in Abb. 12.7 noch einmal grafisch aufbereitet.

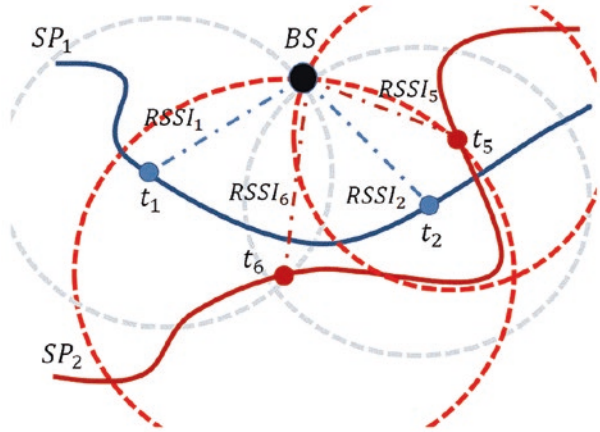

Abb. 12.7 Ergebnis der zweiten Korrelationsstufe: Durch Zusammenlegen der Korrelationen von SP_1 und SP_2 ergibt sich für Google/Apple eine eindeutige Position für die Basisstation. (Quelle: eigene Darstellung in Anlehnung an [3])

12.1.4 Auswirkungen von Swarm-Mapping+

Der geschilderte Ansatz des Swarm-Mapping wird also auf jedem Gerät zu jeder Zeit hinter dem Rücken der Smartphone-Nutzer durchgeführt, und das nicht nur an einem Ort, sondern kontinuierlich weltweit. Aber schon mit unserem sehr einfachen und konstruierten Beispiel lassen sich die Implikationen einer derartigen Komponente zur Positionsbestimmung in den Betriebssystemen Android und Apple zweifelsfrei erahnen. Denn eine solche permanente weltweite Vermessung der drahtlosen Infrastruktur eines Landes oder einer Region öffnet den Raum für weitergehende Spekulationen:

1. Es besteht die Möglichkeit der Korrelation von MAC-Adressen von Smartphone und WLAN-Access-Points und damit die Möglichkeit des Feststellens der Position des Heimat-Access-Points eines jeden Smartphone Nutzers. Wer sich häufig und für längere Zeit an einem WLAN-Access-Point aufhält, wird dort vermutlich zu Hause sein, geschäftlich tätig sein oder dergleichen.
2. Innerhalb von IoT und SmartFactory wird der Einsatz von Industrial-IoT-Geräten immer beliebter. Hierzu zählen auch sogenannte Industrial-IoT-(IIoT-)Gateways, die oftmals mit WLAN-Funktionalität ausgestattet sind. Damit wird jedoch für Google und Apple direkt der Standort von vernetzten Produktionsanlagen sichtbar. Diese Information ist oftmals unkritisch, kann im Zweifelsfall jedoch wertvolle Informationen zum Standort von Produktionsanlagen abfließen lassen.
3. Schließlich kann solch ein Wissen über die aktuellen Positionen von Basisstationen und Access-Points genutzt werden um bösartige Basisstationen und Access-Points bzw. IMSI-Catcher aufzuspüren, die quasi über Nacht platziert worden sind und bei denen sich Smartphonenutzer fälschlicherweise einwählen sollen.

4. Nicht zu unterschätzen sind die geostrategischen Auswirkungen der geschilderten Technologie. Eine weiterführende Diskussion zu den geostrategischen Auswirkungen von Swarm-Mapping+ sowie mögliche Maßnahmen zu deren Eindämmung, findet sich in der Arbeit [3]. Man mache sich klar, dass im militärischen Bereich das Orten gegnerischer Funkstationen eine wesentliche Aufgabe im Rahmen der Fernmelde- und elektronischen Aufklärung war, ist und immer sein wird (auch wenn hier allerdings eher das Orten von Sendemasten für Funktechnologien aus dem militärischen Bereich im Vordergrund stehen dürfte). Die Tatsache, dass moderne mobile Betriebssysteme, die für Geräte des Consumer-Bereichs entwickelt wurden, nun ebenfalls über solche Funktionalitäten verfügen, ist besonders pikant, da hier militärische und nachrichtendienstliche Belange de facto Eingang in die Technologien für den Consumer Bereich gefunden haben.

▶ Swarm-Mapping+ ermöglicht eine recht genaue Approximation der Standorte von Basisstationen und WLAN-Access-Points auf der Basis von am Smartphone anfallenden RSSI-Werten zur Abschätzung der Empfangsfeldstärke drahtloser Kommunikation. Quasi als ‚Nebenprodukt' neben der Positionsbestimmung eines Smartphones ermöglicht diese in mobile Betriebssysteme implementierte intrinsische Lokalisierung (Trilateration am Smartphone) im Zusammenspiel mit der extrinsischen Lokalisierung (Trilaterationen vieler Smartphones auf Servern der mobilen OS-Anbieter) die weltweite und permanente Positionsbestimmung von drahtlosen Infrastrukturkomponenten eines Landes oder einer Region.

12.1.5 Gegenüberstellung der Verfahren zur Positionsbestimmung

Wir wollen uns noch einmal vergegenwärtigen welche der hier geschilderten Verfahren zur Positionsbestimmung und Standortverfolgung auf welcher Technologie beruhen, welche Verfahren eingesetzt werden und insbesondere welche Partei von der Positionsbestimmung profitiert. Für i) das aktive WLAN-Scanning, ii) das Swarm-Mapping sowie iii) Swarm-Mapping+ haben wir die zugrunde gelegte drahtlose Übertragungstechnologie sowie das Mittel der Informationsgewinnung tabellarisch aufgeführt. Des Weiteren erscheint es wissenswert, unter welchen konkreten Vorbedingungen die Bestimmung der Positionsdaten erfolgreich durchführbar ist, also ob beispielsweise das WLAN am mobilen Gerät angeschaltet sein muss oder nicht oder inwieweit ein Verbindungsaufbau erfolgt sein muss oder nicht. Schlussendlich ist von Interesse wer der Nutznießer ist, bei wem also die Positionsdaten anlaufen, und von welchem Gerät die Positionsdaten stammen. Die Tab. 12.1 trägt die wesentlichen bereits herausgearbeiteten Aspekte zu den geschilderten Verfahren daher noch einmal zusammen.

Tab. 12.1 Übersicht zu den verschiedenen Verfahren zur Bestimmung von Positionsdaten und deren Nutznießer. (Quelle: eigene Darstellung)

	Technologie	Verfahren	Erfolgreich	Positions-daten von	Abgedecktes Gebiet	Nutznießer
Aktives WLAN Scanning	WLAN	L2-Probe-anfrage	WLAN ON, Verbindungs-aufbau nicht erforderlich	Mobiles Gerät	WAP-Netz-werk	Access-Point-Anbieter
Swarm Mapping	Zellularer Mobilfunk/ WLAN	RSSI (Trilateration)	Auch bei WLAN OFF	Mobiles Gerät	Weltweit	Mobile OS-Anbieter
Swarm Mapping+	Zellularer Mobilfunk/ WLAN	RSSI (Trilateration)	Auch bei WLAN OFF	WLAN-Ac-cess-Point/ Sendestation	Weltweit	Mobile OS-Anbieter

12.1.6 Auskunftsansprüche und datenschutzrechtliche Erwägungen

Die Positionsbestimmung und die Standortverfolgung von Nutzern digitaler Geräte mit verschiedensten Übertragungstechnologien ist also für eine ganze Reihe von Parteien und Diensteanbietern möglich. Zwar unterscheidet sich für die Anbieter eines WAP-Netzwerkes, für einen Telekommunikationsanbieter, besagte Hersteller mobiler Betriebssysteme Android und iOS oder auch für Anbieter von Bluetooth- und NFC-Anwendungen der Abdeckungsbereich der Tracking-Region sowie die Genauigkeit der anfallenden Positionsdaten teils erheblich, ein Sachverhalt aber bleibt bei allen Anwendungsfällen immer der gleiche: Es fallen massiv Log-Dateien mit personenbezogenen Daten an. Für die Anbieter öffentlicher Telekommunikationsdienste besteht zudem die Pflicht der Vorratsdatenspeicherung für jedes Telefonat [4], also für jeden Anruf insbesondere die Speicherung der IMEI des Anrufenden sowie des Angerufenen. Aber auch hier, wie auch bei den anderen oben genannten Diensteanbietern fallen IP-Adressen und/oder oder MAC-Adressen der Nutzer an. Dass es sich bei all diesen Identifikatoren um personenbezogene Daten handelt, also laut Bundesdatenschutzgesetz (BDSG) *„Einzelangaben über persönliche oder sachliche Verhältnisse einer bestimmten oder bestimmbaren natürlichen Person“,* wurde vielerorts debattiert, beispielsweise in [4]. Dabei wird üblicherweise argumentiert, dass die ‚Bestimmbarkeit einer natürlichen Person‘ häufig durch das Ablegen von Identifikatoren wie IMEI, IP-Adresse oder MAC gegeben ist. Während die ‚sachlichen Verhältnisse‘ im Zusammenhang mit Log-Dateien und deren Zugangsinformationen mobiler Nutzer durch die Tatsache gegeben ist, dass hierüber die Nähe der natürlichen Person zu einem bestimmten Zeitpunkt zu einer Sendestation festzumachen ist. Solche Datensätze mobiler Zugangsdaten und diverse Identifikatoren sind nicht nur für die eigenen Geschäftsmodelle, sondern eben auch für

Strafverfolgungsbehörden von hohem Interesse, wobei der Paragraph § 113c des Telekommunikationsgesetzes (TKG) Auskunft über die Regelung des Zugriffs durch Strafverfolgungsbehörden gibt. So führen Ronald Petrlic und Christoph Sorge in [4] aus, dass unter Berufung auf eine gesetzliche Bestimmung eine Strafverfolgungsbehörde (Staatsanwaltschaft) zur Verfolgung von besonders schweren Straftaten auf die Daten zugreifen darf. Gleiches gilt für die Gefahrenabwehrbehörden der Länder (Polizei). Eine Aussage im Telemediengesetz (TMG) zur Anfrage, ob der Bundesnachrichtendienst berechtigt, ist von Diensteanbietern die Herausgabe personenbezogener Daten zu verlangen, wird in einer Gesetzesbegründung (Bundestags-Drucksache 16/3078) laut [4] beantwortet mit *„… Die Vorschrift besagt, dass Diensteanbieter aus der Aufgabenerfüllung im Bereich der Strafverfolgung sowie der genannten Behörden erwachsende Auskunftsansprüche nicht aus datenschutzrechtlichen Erwägungen zurückweisen können …“*.

Der Jurist Dr. Uwe Ewald konkretisiert dies:

> „Ermittlungsbehörden berühren bei Zugriffen auf personenbezogene Daten zur Positionsbestimmung und Standortverfolgung die Vertraulichkeit und Integrität informationstechnischer Systeme und damit den verfassungsrechtlich garantierten Schutz aus dem Allgemeinen Persönlichkeitsrecht (Art. 2 Abs. 1 i. V. m. Art. 1 Abs. 1 GG). Mit Datenschutzgrundverordnung und der Justiz-und-Inneres-(JI-)Richtlinie wurden entsprechende EU-Regelungen zum Schutz persönlicher Daten geschaffen. Die gesetzliche Grundlage zur Legitimation von Eingriffen werden für die Strafverfolgungsbehörden in der Strafprozessordnung, z. B. in § 100 g StPO „Erhebung von Verkehrsdaten“ mit Verweis auf § 96 Absatz 1 des Telekommunikationsgesetzes sowie § 2a Absatz 1 des Gesetzes über die Errichtung einer Bundesanstalt für den Digitalfunk der Behörden und Organisationen mit Sicherheitsaufgaben geregelt. Im Bereich der Gefahrenabwehr erfolgt die gesetzliche Regelung im Polizeirecht des Bundes und der Länder.“

Durch diese juristischen Gegebenheiten ergibt sich aber in gleich mehrfacher Hinsicht ein datenschutzrechtliches Dilemma:

a) Die Behörde möchte dem Anbieter eines Telekommunikationsnetzes, eines WAP-Netzes oder dergleichen nicht zwingend mitteilen, nach welchen Verdächtigen konkret gefahndet wird.
b) Gleichzeitig sieht sich der Anbieter und Sammler der Datensätze mit mobilen Zugangsdaten in einem Dilemma. Auf der einen Seite möchte und muss er den Behörden Auskunft erteilen (s. o.), gleichzeitig fühlt er sich seinen Nutzern und Kunden zumindest moralisch verpflichtet, Dritten keinen Einblick in den vollständigen Datensatz zu gewähren. Denn dies würde bedeuten, einen Großteil seiner unbescholtenen Nutzer einem enormen ‚Beifang‘ auszusetzen.

Diese Vorbemerkungen sollten, rein akademisch betrachtet, die beiden ersten Varianten der praktischen Umsetzung ausschließen, gleichwohl dürfte die zweite Variante der Herangehensweisen die wohl aktuell praktizierte sein:

Variante 1 Die Behörde übergibt dem Diensteanbieter die Identifikatoren derjenigen natürlichen Personen, nach denen gefahndet wird, damit der Diensteanbieter die Suche auf seinen Log-Dateien durchführen kann. Dieses Vorgehen ist für die Behörden nachteilhaft und vermutlich nicht tolerierbar, da dies einen Erkenntnisgewinn für den Diensteanbieter gäbe (dies wäre u. U. noch vertretbar bei staatlichen Diensteanbietern, aber sicherlich nicht mehr bei den vielen privaten Anbietern von WAP-Netzen, NFC-Anwendungen und dergleichen).

Variante 2 Der Diensteanbieter übergibt die relevanten Log-Dateien an die Behörde, die dann die Suche nach den fraglichen natürlichen Personen eigenständig durchführen kann. Damit erhält sie aber auch ebenfalls Einblick in die Aufenthaltsorte eine Vielzahl von unbescholtenen, nicht unter Strafverdacht stehenden Bürgern, die sich zur fraglichen Zeit zufällig ebenfalls im Sendebereich der Zelle aufgehalten haben.

Variante 3 Eine weitere Variante wäre die Anonymisierung der personenbezogenen Datensätze derart, dass mit einer solchen technischen Maßnahme nur noch ein sehr geringes Risiko einer Re-Identifizierung bestünde. Bitte beachten Sie, dass diese Forderung die einfach durchzuführende und insbesondere statische Pseudonymisierung der Datensätze ausschließt, da diese oftmals mit einfachen Mittel auszuhebeln ist und damit zur Aufdeckung der Identität einer natürlichen Person führen kann.

12.1.7 Auskunftsansprüche mit technischem Datenschutz

Setzt der Betreiber technische Maßnahmen ein, um Datenschutz in seine Systeme zu integrieren, dann spricht man von technischem Datenschutz, oder aber auch von *Privacy-by-Design.* Letzterer Begriff findet sich auch in der Datenschutz-Grundverordnung (DSGVO) wieder. Eine mögliche Umsetzung der im vorherigen Abschnitt skizzierten dritten Variante mit dem Ziel einer Anonymisierung der Log-Dateien mit Identifikatoren von natürlichen Personen wollen wir nun ein wenig genauer erörtern. Gleichzeitig werden auch die Listen der Verdächtigen anonymisiert. Dabei ist es das konkrete Ziel, Behörden mit Auskunftsansprüchen eine Post-mortem Positionsverfolgung von Verdächtigen zu ermöglichen, ohne dass nicht unter Verdacht stehende Nutzer und Kunden besagtem ‚Beifang' ausgesetzt wären.

Der technische Datenschutz kennt mit *k-Anonymität* [5], *I-Diversität* oder der *Differential Privacy* [6] gleich eine Reihe von Ansätzen zur Anonymisierung von personenbezogenen Datensätzen. Ohne auf diese Ansätze an dieser Stelle einzugehen, bleibt jedoch festzuhalten, dass man oftmals nicht umhinkommen wird, eine passgenaue Lösung für jeden Anwendungsfall zu betrachten. Der Doktorand Louis Tajan hat in [7] genau dies für die obige Problemstellung getan und einen Ansatz auf Basis von Bloom-Filtern vorgeschlagen und evaluiert. Der Jurist Dr. Uwe Ewald, spezialisiert auf

Datenanalyse in Ermittlungsverfahren und mit dem Themenbereich der gerichtsverwertbaren forensischen Analyse vertraut, merkt zu dem nun geschilderten Ansatz an, dass er insbesondere deshalb interessant sein könnte, da dieser trotz einer Umsetzung eines technischen Datenschutzes noch Auskunft auf die für die Praxis wesentliche Fragestellung *„Welche Gruppe von Verdächtigen war in der Nähe?“* geben kann.

Die genaue mathematische Analyse des Verfahrens kann unter [8] eingesehen werden, an dieser Stelle wollen wir nur die prinzipielle Machbarkeit des Ansatzes erörtern sowie auf potenzielle Fallstricke hinweisen. Wie wir es bereits in Abschn. 2.4.8 angedeutet haben, wurde die ursprüngliche von Burton Howard Bloom vorgeschlagene Datenstruktur in leicht abgewandelter Form betrachtet, indem nun ein Message Authentication Code, konkret ein HMAC anstelle von k unterschiedlichen Hashfunktionen verwendet wird.

Es werden, wie in Abschn. 2.4.8 zu Bloom-Filtern bereits geschildert, zwei Bloom-Filter erzeugt. Ein Filter bei der Behörde und ein weiterer Filter beim Diensteanbieter:

Bei der Behörde Der Datenstruktur BF_V liegt die Menge V der Liste aller Verdächtigen zugrunde.

Beim Diensteanbieter Die Datenstruktur BF_L ist aus der Menge L hervorgegangen, die die Identifikatoren der Datensätze der mobilen Zugangsdaten zu einer Sendestation enthält

Damit die beiden Parteien ihre Bloom-Filter passgenau erzeugen können, müssen mit dem Zeitpunkt der Umsetzung der Auskunftsansprüche allerdings eine Reihe von Parametern zwischen den Parteien festgelegt werden: die Länge m beider Bloom-Filter, wobei $m = |BF_V| = |BF_L|$, die Anzahl der HMAC-Berechnungen k sowie die für die HMAC-Berechnung zu verwendenden Schlüssel K_i mit $i = 1, \ldots, k$. Dabei hat es gerade die Festlegung der Dimension beider Bloom-Filter in sich. Denn diese ist für ein vorteilhaft justiertes System festzulegen in Abhängigkeit von den Größenordnungen der Mengen V und L. Solche Vorabbetrachtungen sind zwingend erforderlich, um zwei Bloom-Filter zu erzeugen, die ein ausgewogenes Maß an *Datenschutz* (engl: *privacy*) und *Korrektheit* bieten. Dabei sollen hinsichtlich der Privacy-Anforderung nicht nur der Inhalt, also die Elemente, „gut“ anonymisiert werden, sondern es ist darüber hinaus auch das erklärte Ziel, die Kardinalität der Mengen V und L zu verbergen. Denn gerade bei kleineren Mengen, und dies dürfte zumindest für die Menge der Verdächtigen V der Fall sein, könnten sich hierüber Anknüpfungspunkte für Rückschlüsse auf Elemente der Menge geben. Die Anonymisierung der Inhalte der Mengen ergibt sich durch die Anwendung des HMAC mit geheim zu haltenden Schlüsseln und der Tatsache, es hier mit einer Einwegfunktion (siehe Abschn. 2.4.2 und 2.4.3) zu tun zu haben. Sollen einem Angreifer idealerweise auch keinerlei Rückschlüsse von einem Bloom-Filter auf die Kardinalität der repräsentierten Menge ermöglicht werden, so sind weitere Überlegungen

erforderlich. Würde dieser die Zahl k kennen und wäre darüber hinaus die Länge m des Bloom-Filters unverhältnismäßig groß gewählt in Bezug zur Anzahl der Elemente der Menge, dann könnte der Angreifer durch einfaches Zählen aller 1-er-Bits und das anschließende Teilen dieses Wertes durch k bereits eine gute Näherung an die Kardinalität der Menge erhalten. Dies liegt daran, dass in solch einem Fall der Bloom-Filter dünn besetzt ist, also recht wenig 1er- im Vergleich zu 0er-Einträgen zu zählen sind. Oder noch ein wenig konkreter: Um die Kardinalität der Menge zu verbergen, ist es von Vorteil, wenn bei der Erstellung eines Bloom-Filters und dem Eintragen einzelner 1er-Bits für ein konkretes Element es zu Überlagerungen kommt. Es wird also eine 1 an einer Stelle im Bloom-Filter eingetragen, an der bereits ein 1er-Bit eingetragen worden ist. Solche überlagernden Bits sorgen dafür, dass ein Angreifer den oben geschilderten einfachen Angriff auf die Kardinalität der Menge nicht mehr so ohne Weiteres durchführen kann. Sie haben allerdings auch einen Nachteil, denn sie führen zu *False-Negatives:* Es kann also mit einer sehr geringen Wahrscheinlichkeit vorkommen, dass ein Element doch nicht zu der Menge gehört, obwohl der Bloom-Filter-Test das Gegenteil behauptet. Will man also die Wahrscheinlichkeit von False-Negatives möglichst klein halten, ist es ratsam, den Wert für m groß zu wählen. Dies geht aber auf Kosten der Verheimlichung der Kardinalität der Menge, sodass es notwendig ist, hier einen guten Kompromiss zu finden. Einzelheiten zur Wahl von m finden sich in [8].

Wir halten also fest, dass der Diensteanbieter und die Behörde jeweils ihre Datenstruktur BF_V und BF_L erstellt haben und nur diese Datenstrukturen an eine Prüfstelle übergeben. Sowohl k als auch die Schlüssel K_i mit $i = 1, \ldots, k$ dürfen der Prüfstelle nicht bekannt sein. Will man diese Prüfstelle einer Angreiferkategorie zuordnen, so wäre diese am besten zu beschreiben als *„ehrlich, aber neugierig“* (engl: *honest-but-curious*). Dies bedeutet nichts anderes, als dass man darauf vertrauen kann, dass diese Partei die ihr aufgetragenen Tätigkeiten vertrauensvoll und gewissenhaft durchführt. Man könnte jedoch nicht vollständig darauf vertrauen, dass sie eventuelle Informationen, die sich aufgrund dieser Tätigkeit offenbaren würden, nicht doch auf die eine oder andere Weise ausnutzen würde.

Bevor wir nun die an der Prüfstelle anfallenden einzelnen Arbeitsschritte erörtern, ist in Abb. 12.8 die mögliche Protokollabfolge zum behördlichen Auskunftsanspruch mit technischem Datenschutz aufgezeigt.

Bei der Prüfstelle Entscheidend für die Durchführbarkeit des Ansatzes ist die Möglichkeit des Testens der Mengen L und V auf Elementfremdheit, ob also gilt, dass $V \cap L = \{\}$ ist oder eben nicht. Dies soll validiert werden, ohne dass der prüfenden Instanz diese beiden Mengen vorliegen. Die Prüfstelle erhält also die Datenstrukturen BF_V und BF_L und berechnet aus diesen beiden Bloom-Filtern die Datenstruktur $BF_{L \cap V}$, also wiederum einen Bloom-Filter. Hierzu verwendet sie die bitweise logische Operation $\wedge$:

$$BF_{V \cap L} = BF_V \wedge BF_L$$

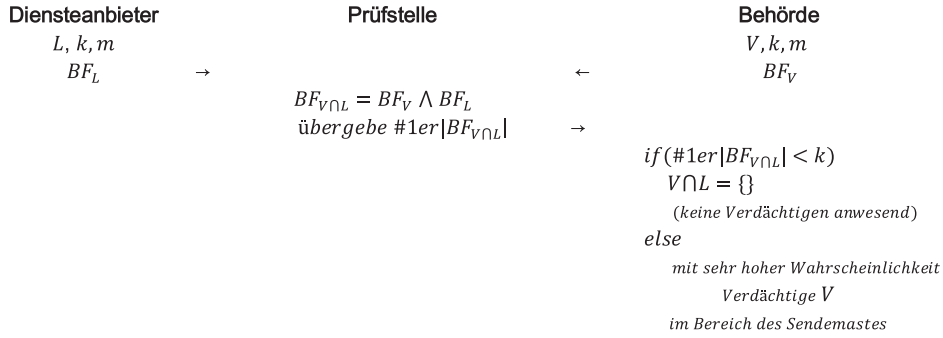

Abb. 12.8 Mögliche nicht interaktive Protokollabfolge zum behördlichen Auskunftsanspruch bei Strafverfolgung mit technischem Datenschutz, basierend auf Bloom-Filtern. (Quelle: eigene Darstellung)

Die Berechnung von $BF_{V\cap L}$ für zwei beispielhafte Bloom-Filter BF_V und BF_L führt zu dem Bloom-Filter $BF_{V\cap L}$:

$$\begin{aligned} BF_L &= 1001\ldots0010 \\ \underline{\wedge BF_V} &= \underline{1011\ldots1010} \\ BF_{V\cap L} &= 1001\ldots1010 \end{aligned}$$

Anschließend ermittelt die Prüfstelle durch einfaches Zählen der Anzahl der in $BF_{V\cap L}$ enthaltenen 1er-Bits den Wert $\#1|BF_{V\cap L}|$ und übermittelt diesen an die Behörde. Die Behörde kann nun validieren, ob $V\cap L = \{\}$ oder nicht. Denn ist der Wert $\#1|BF_{V\cap L}|$ kleiner als k, so gilt, dass die Verdächtigen sich nicht im Bereich des Sendemastes aufgehalten haben. Zählte die Prüfstelle jedoch mehr als k 1er-Bits in $BF_{V\cap L}$, dann haben sich alle Verdächtigen mit einer sehr großen Wahrscheinlichkeit in der Nähe des Sendemastes aufgehalten. Dieser Test auf Elementfremdheit zweier Mengen, ohne in Besitz ebendieser zu sein, führt somit leider mit einer wenn auch nur sehr geringen Wahrscheinlichkeit zu einem *False-Negative*-Urteil der Prüfstelle. Die Prüfstelle behauptet, dass die Mengen V und L nicht leer sind, tatsächlich sind sie es aber doch. In [8] sind beispielhafte Wertekombinationen für das Eintreten eines *False-Negative*-Urteils aufgeführt. So würde bei zwei Mengen $|L| = 2 \cdot 10^2$ und $|V| = 10^4$, $k = 1468$ und $m = 9,47 \cdot 10^9$ die Wahrscheinlichkeit eines *False-Negative*-Urteils bei ~0 liegen.

Lassen Sie uns abschließend noch einzelne Denkanstöße zu den Fähigkeiten und Limitierungen des hier vorgestellten Verfahrens vornehmen:

Ad1 Die Prüfstelle erhält keinerlei Wissen zum Ergebnis von $V\cap L$, da sie insbesondere k nicht kennt. Sie weiß nach dem Test also weder, nach wem bzw. welcher Gruppe konkret gefahndet wird, noch, ob die Suche neue Hinweise zu den Verdächtigen gegeben hat.

Ad2 Auch, wenn sich die Verdächtigen zu unterschiedlichen Zeitpunkten in der Nähe des Sendemastes eingefunden haben, so ist der hier beschriebene Ansatz durchführbar. Die Log-Daten L sind dann über ein größeres Zeitfenster anzuwenden.

Ad3 Allerdings wird die Prüfung auch dann das Ergebnis ‚keine Verdächtigen anwesend' liefern, wenn sich eine Untermenge $\tilde{V} \subset V$ in der Nähe des Sendemastes befunden hat. Dies gilt es zu beachten und eine Suche müsste gegebenenfalls verfeinert werden.

▶ Die Verwendung von mit einem HMAC erzeugten Bloom-Filtern BF_L und BF_V als repräsentierende Datenstrukturen für die Log-Dateien L mobiler Telekommunikationsanbieter und für eine Liste V von Verdächtigen kann dazu dienen, Behörden mit Auskunftsansprüchen eine Post-mortem-Positionsverfolgung zu ermöglichen, ohne dass nicht unter Verdacht stehende Nutzer einem ‚Beifang' ausgesetzt wären. Um ein ausgewogenes Maß an *Datenschutz* und *Korrektheit* (*False-Negatives* sind möglich) zu gewährleisten, ist im Vorfeld einer Prüfung insbesondere die Länge $m = |BF_V| = |BF_V|$ beider Bloom-Filter sorgfältig zu wählen. Hat sich nur eine Untermenge $\tilde{V} \subset V$ Verdächtiger in der Nähe des Sendemastes aufgehalten, so ist das Ergebnis ‚keine Verdächtigen anwesend'. Hier müsste die Größe L der Log-Datei beispielsweise auf die angefallenen Daten benachbarter Sendemasten ausgedehnt werden.

Diejenigen Leser, die nach der Beschreibung dieses Verfahrens ein weitergehendes Interesse an Fragestellungen rund um eine forensische Analyse von Daten mobiler Geräte und deren Verwertbarkeit vor Gericht haben, seien auf die Website des EU-Projektes ForMobile [9] aus dem *Horizon 2020 Programme* der Europäischen Kommission verwiesen: Das Projekt trägt den Untertitel *From Mobile Phones to Court* und beschäftigt sich mit der Entwicklung standardisierter Prozesse für eine forensische Analyse der an mobilen Geräten anfallenden Daten und einer vor Gericht verwertbaren Nutzung ebendieser. Das Konsortium setzt sich aus 19 Partnern aus 15 Ländern zusammen. Es läuft noch bis ins Jahr 2022.

12.2 Zusammenfassung

Wir haben uns im Laufe dieses Kapitels mit unterschiedlichen Verfahren zur Lokalisierung, Positionsbestimmung und Standortverfolgung beschäftigt. Offensichtlich kann und wird jede der in einem Smartphone anzutreffenden drahtlosen Übertragungstechnologien genutzt um die Position eines Gerätes zu bestimmen oder seinen Standort zu verfolgen. Im Falle der Übertragungstechnologie WLAN erfolgt die Suche nach einem WLAN-Access-Point mehrheitlich mithilfe von aktivem WLAN-Scanning,

was zur Folge hat, dass dem Betreiber des Access-Points oder des WAP-Netzwerkes die Positionsdaten des mobilen Gerätes vorliegen. Dass dies auch ohne tatsächlich erfolgten Verbindungsaufbau möglich ist, dürften viele Nutzer dieser Technologie nicht wissen oder billigend in Kauf nehmen. Anschließend haben wir die Möglichkeiten der Positionsbestimmung und der Standortverfolgung mittels Swarm-Mapping und Swarm-Mapping+ für mobile Geräte, die das Betriebssystem Android oder iOS nutzen, erörtert. Wir haben erfahren, dass eine Positionsbestimmung in zwei sich ständig wiederholenden Phasen erfolgt und dadurch Schritt für Schritt verfeinert wird: in der intrinsischen Lokalisierung, die auf dem mobilen Gerät selbst stattfindet, und der extrinsischen Lokalisierung, die auf einem größeren Datensatz aller mobilen Geräte in der Nähe einer bestimmten Sendestation auf Servern der Firmen Google und Apple erfolgt. Sowohl die Phase der intrinsischen Lokalisierung als auch die Phase der extrinsischen Lokalisierung erfolgen insbesondere unter Verwendung der Received Signal Strength Indication (RSSI) und der mutmaßlichen Nutzung von Verfahren der Trilateration. Dabei kommen die Verfahren Swarm-Mapping und Swarm-Mapping+ sowohl für die drahtlose Übertragungstechnologie WLAN also auch in Mobilfunkzellen zum Einsatz. Wir haben erörtert, welche unterschiedlichen Regionen die beschriebenen Verfahren zur Lokalisierung und Positionsbestimmung abdecken und wer die Nutznießer derartiger Standortinformationen digitaler Geräte sind. Neben den datenschutzrechtlichen Bedenken, die bei einer derartigen Anhäufung von Standortinformationen von Identifikatoren zu mehrheitlich natürlichen Personen zwangsläufig ins Auge fallen, haben wir im Falle von Swarm-Mapping+ auch auf die geostrategische Bedeutung um das alleinige Wissen der weltweiten Standorte von Sendemasten und WLAN-Access-Points hingewiesen. Im Zusammenhang mit den datenschutzrechtlichen Erwägungen von Anbietern eines WAP-Netzwerkes, aber natürlich auch von Telekommunikationsanbietern haben wir auf die gesetzlich geregelten Auskunftsansprüche von Gefahrenabwehrbehörden und Strafverfolgungsbehörden o. ä. hingewiesen was oftmals für die Diensteanbieter zu einem datenschutzrechtlichen Spannungsfeld führen dürfte. Wir haben einen beispielhaften Lösungsansatz durch technischen Datenschutz (Privacy-by-Design) erörtert, der Behörden mit Auskunftsansprüchen eine Post-mortem Positionsverfolgung von Verdächtigen ermöglichen würde, ohne dass die Positionsdaten nicht unter Verdacht stehender Nutzer als ‚Beifang' anfallen würden. Hierzu haben wir als Datenstruktur Bloom-Filter verwendet, mit denen endliche Mengen kompakt dargestellt werden können, und diese für unsere Zwecke angepasst. Eine denkbare nicht interaktive Protokollabfolge zum behördlichen Auskunftsanspruch bei einer Strafverfolgung würde dann nicht mehr auf Log-Dateien und Listen von Verdächtigen operieren, sondern auf den Bloom-Filtern, welche die Log-Dateien des Diensteanbieters und Liste der Verdächtigen der Behörde in einer datenschutzkonformen Darstellung repräsentieren.

Literatur

1. https://www.blm.de/files/pdf1/140512_Location-based_Services_Monitor_2014.pdf. Zugegriffen: 26. Apr. 2019
2. Dhein, A., Grimm, R.: Standortlokalisierung in modernen Smartphones. Informatik-Spektrum **40**(3), 245–254 (2017)
3. Westhoff, D., Zeiser, M.: Measuring the world – how the smartphone industry impacts cyber deterrence credibility. Int. J. Cyber Warfare Terror (IJCWT) **8**(2), 1–16 (2018)
4. Petrlic, R., Sorge, C.: Datenschutz – Einführung in technischen Datenschutz, Datenschutzrecht und angewandte Kryptographie. Springer Vieweg, Wiesbaden (2017)
5. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. Uncertainty **10**(5), 557–570 (2002)
6. Dwork, C.: Differential privacy. In: Automata, Languages and Programming, LNCS 4052, S. 1–12. Springer (2006); CALP: International Colloquium on Automata, Languages, and Programming Automata, Languages and Programming 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II Editors (view affiliations) Michele BugliesiBart PreneelVladimiro SassoneIngo Wegener
7. Tajan, L., Westhoff, D.: Retrospective tracking of suspects in GDPR conform mobile access networks datasets. CECC 2019, Proceedings of the 3rd Central European Cybersecurity Conference (2019)
8. Tajan, L., Westhoff, D., Armknecht, F.: Private set relations with bloom filters for outsourced SLA validation. IACR Cryptol. ePrint Archive **2019**, 993 (2019)
9. Formobile, EU Project. https://formobile-project.eu/. Zugegriffen: 18. Nov. 2019

Teil IV
Fazit und Ausblick

Nicht technische Ursachenforschung

13

Jenseits aller Risikoabschätzungen der aufgezeigten Verwundbarkeiten verschiedener drahtloser Kommunikationsprotokolle und mobiler Betriebssysteme ist es zum Ende dieses Buches vermutlich hilfreich, sich auf die Spurensuche nach den übergeordneten Gründen für die von den Sicherheitsforschern identifizierten Schwachstellen und Verwundbarkeiten zu begeben. Wir wollen also nach der in diesem Buch betriebenen Bestandsaufnahme ein wenig Ursachenforschung betreiben. Meiner Ansicht nach sind die Gründe für die aufgezeigten teilweise haarsträubenden Sicherheitslücken tatsächlich nicht immer nur technischer Natur. Folgende Aspekte haben nach Meinung des Autors ebenfalls teils gravierenden Einfluss auf das Auftreten von Verwundbarkeiten und Schwachstellen in Übertragungsprotokollen oder ganzer IT-Systeme:

‚Unter der Annahme, dass'
Um seinen Beitrag innerhalb der Forschungsgemeinschaft gegenüber anderen Arbeiten abzugrenzen und um darauf hinzudeuten, auf welchen Grundlagen die eigene Sicherheitsarchitektur fußt, hat es sich bei akademischen Veröffentlichungen im Bereich der IT-Sicherheit bewährt, dies mit den Worten ‚Unter der Annahme, dass die Komponente x sicher ist, ist unser Ansatz zur Lösung von y sicher' zur Geltung zu bringen. Dies ist, wie gesagt, eigentlich bewährte akademische Praxis. Allerdings birgt diese Gepflogenheit auch Gefahren in sich, und zwar immer dann, wenn eine akademische Arbeit als Grundlage in einen Standard eingeht, um darauf aufsetzend den Herstellern eine Spezifikation für ein Softwareprodukt an die Hand zu geben. Denn mit jeder Lage an weiteren ‚Meta-Dokumenten' und Editoren von Drafts und Spezifikationen, welche die ursprüngliche Arbeit nur noch referenzieren, steigt die Gefahr, dass bis zur tatsächlichen Umsetzung des Standards diese ‚Wenn, dann'-Beziehung der im ursprünglichen Dokument aufgezeigten Sicherheitsabhängigkeit nicht mehr hinterfragt wird. In der Zwischenzeit kann sie sich aber als unvollständig erwiesen haben oder sich als zwar

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_13

notwendige, jedoch nicht als hinreichende Bedingung herausgestellt haben. Ein weiterer, eher profaner Grund kann auch sein, dass zwischenzeitlich die Sicherheit der Komponente x maßgeblich aufgrund eines neuen Angriffsmusters oder erhöhter Rechenleistung reduziert ist. Prominentes Beispiel hierfür ist die Verwendung von Hashfunktionen (Komponente x) zur Erzeugung von digitalen Signaturen beziehungsweise von Zertifikaten (Komponente y). Doch solche sich einmal in die Spezifikationen eingenisteten fehlerhaften Abhängigkeiten sind sehr schwer auszumerzen. Denn das Verabschieden einer Anpassung der Spezifikation ist das eine, die jeweilige Anpassung in den Produkten eine ganz andere Geschichte, insbesondere bei solchen Produkten, die schon auf dem Markt oder gar langjährig im Einsatz sind.

‚Moving-Target‘
IT-Sicherheit ist, um es ein wenig martialisch auszudrücken, ein ‚Moving Target‘. Jeder Exploit kursiert erst einmal als Verwundbarkeit, die nur einer sehr kleinen Zahl von Experten bekannt ist. Danach wird aus ihr (hoffentlich) eine öffentlich bekanntgegebene Verwundbarkeit mit (hoffentlich) den ersten Patches hierzu sowie Anpassungen von Spezifikationsdokumenten und das Aufspielen entsprechender Versionen bzw. OS-Varianten. Innerhalb dieses Zeitfensters, das sich oftmals als viel zu groß erweist, sind Geräte und Systeme über diesen konkreten Exploit angreifbar. Typischerweise rollt dann schon die nächste Welle mit einer weiteren geheimen Verwundbarkeit an und das Spiel beginnt von neuem. Oder anders ausgedrückt: Ein vollständig sicheres Gesamtsystem, das gehärtet ist gegen alle aktuell existierenden geheimen Verwundbarkeiten, ist nach dieser Betrachtungsweise wohl eher die Ausnahme als die Regel.

‚Zu komplex‘
Die Komplexität von Protokollen, Protokoll-Stacks, Betriebssystemen, deren vielfältigen Umsetzungen sowie deren Zusammenwirken ist in all ihren Sonderfällen und Schattierungen schier nicht mehr nachvollziehbar, weder unter Verwendung statischer noch dynamischer Analysewerkzeuge.

Zustandsautomaten in Spezifikationen
Oftmals enthalten die einzelnen von den Standardisierungsgremien verabschiedeten Spezifikationen keine formale Beschreibung eines Zustandsautomaten für die Ausgestaltung der zu realisierenden Client-Prozesse und Server-Prozesse. Anstelle dessen wird ein Pseudocode verwendet, aus dem zu entnehmen ist, wie eine Nachricht aufgebaut ist, aber nicht, *wann* genau diese zu bearbeiten ist und welche Auswirkung dies auf den Übergang einer Entität in einen anderen Zustand hat. Im Falle von WPA/WPA2 hat die vereinfachte Darstellung eines Zustandsautomaten für den Client den KRACK-Angriff begünstigt.

‚Quick-and-Dirty‘
Man sollte keinesfalls die Lebenszeit von *‚Quick-and-Dirty‘*-Umsetzungen unterschätzen. Deren Auswirkungen sind vermutlich genau an solchen Stellen umso schwerwiegender und häufiger anzutreffen, an denen der Programmierer aufgrund einer laxen oder unvollständig formulierten Aussage zu speziellen Parametern innerhalb der Spezifikation mit deren konkreter Ausgestaltung allein gelassen wird.

Interessenkonflikte
Innerhalb von Standardisierungsgremien gibt es reichlich Interessenkonflikte zwischen den eingebundenen Parteien, sodass die in einer Spezifikation letztendlich niedergeschriebene Variante eines Vorgehens nicht immer zwangsläufig auch die sicherste sein muss. Die einen möchten, dass von dem Standard auch leistungsschwache eingebettete Geräte profitieren, auf denen gewisse erforderliche Randbedingungen nicht zur Verfügung stehen, die anderen erwarten, dass ein abwärtskompatibles Produkt entsteht, selbst dann noch, wenn es kompatibel zu Produkten mit veralteten, unsicheren Sicherheitslösungen ist. Wieder andere argumentieren, dass eine erhöhte Sicherheit zu Lasten der Nettodatenrate auf der Übertragungsstrecke geht, und fürchten für den Consumer-Bereich die Abwanderung von Kunden zu anderen Anbietern. Und wieder andere Interessengruppen fürchten Exportbeschränkungen für den amerikanischen Markt bei zu großen Schlüssellängen. Man muss gar nicht fabulieren, wenn man über *‚Hidden agendas‘* spricht, bei denen Interessengruppen bewusst schwache Sicherheitsparameter in Standarddokumente eingehen lassen, die von typischen Angreifern kaum zu brechen sind, aber von den Institutionen, deren Interessen jene in den Arbeitsgruppen vertreten.

Patente, Open-Source, Security-by-Obscurity
Es gibt verschiedenste Ebenen, Algorithmen und Implementierungen lizenzrechtlich zu schützen (Patente) und verwendbar zu machen, ohne jedoch deren konkrete Arbeitsweise preis zu geben (Security-by-Obscurity) oder aber bewusst öffentlich zu gestalten, in dem Bewusstsein, dass gerade bei Code, der Auswirkungen auf die IT-Sicherheit von Produkten hat, es sehr hilfreich ist, wenn die gesamte Community diesen auf Schwächen untersucht und bedarfsweise weiterentwickelt. Oftmals ist es aber so, dass Sicherheitsalgorithmen, die durch ein Unternehmen patentiert sind, gerade dann nicht verwendet werden. Hingegen eingesetzte sind derartige Algorithmen, die zum Einsatz kommen und deren Funktionalität nicht veröffentlicht wurde, oftmals über die Zeit de-obfuskiert worden und haben spätestens dann zu einer Sicherheitslücke beigetragen.

Analysewerkzeuge
Das Arsenal digitaler Werkzeuge sowie zur Verfügung stehender Bibliotheken wird immer leistungsstärker und allumfassender. Neben klassischen Netzwerkanalyse-Werkzeugen für aktives Netzwerk-Capturing (airsnort), Protokolltests mit netcat und nmap, dem Test von Webanwendungen (Burp Suite), Werkzeugen für Fuzzing und für die Entwicklung von Exploits (Kali Linux, Scapy) sowie für das Netzwerk-Spoofing

(Ettercap) wurden auch eine Reihe von Werkzeugen für Pentests und Reverse Engineering im mobilen Bereich entwickelt: Neben adb sind beispielsweise zur Analyse unter Android hier dex2jar, enjarify, baksmali, JD-GUI oder Codeinspect aufzuführen. Frida ist das Werkzeug zur Analyse und Manipulation von Android- und iOS-Anwendungen.

Berichterstattung zu Verwundbarkeiten
Wenn Sicherheitsforscher neue noch nicht dokumentierte Verwundbarkeiten in Protokollen, Software oder Produkten entdecken, so sollte die Abfolge der Offenlegung dieser Verwundbarkeiten idealerweise in einer gewissen Reihenfolge geschehen. Diese, unter der Bezeichnung verantwortungsbewusste Bekanntmachung (engl. *responsible disclosure*) bekannte Abfolge sieht beispielsweise die folgende zeitlich aufeinander aufbauende Abfolge vor:

T1: Entdeckung der Verwundbarkeit
T2: Benachrichtigung des Herstellers
T3: Hersteller bestätigt die Verwundbarkeit
T4: Anfrage einer Common Vulnerabilities and Exposures (CVE) bei der MITRE
T5: Hersteller gibt Update und Sicherheitsratschlag für CVE-Jahr-Nummer frei

Antwortet der Hersteller nicht binnen einer Frist von typischerweise 30 Tagen auf die Benachrichtigung der Sicherheitsexperten, so können die Entdecker der Verwundbarkeit diese bekannt machen. Gleiches gilt für den Fall, dass der Hersteller nicht binnen einer Frist von typischerweise 90 Tagen das Sicherheitsupdate sowie den Sicherheitsratschlag für seine Kunden bereitstellt. Diese zeitliche Abfolge offenbart, wie lange Verwundbarkeiten in IT-Produkten existieren, selbst dann noch, wenn sie entdeckt wurden und der Hersteller benachrichtigt wurde. Die MITRE zur Vergabe von CVE Nummern ist eine Organisation in den Vereinigten Staaten.

IT-Sicherheit ist ein Geschäft
Wer es noch nicht gemerkt haben sollte: IT-Sicherheit ist ein Geschäft! Nur die ständige Abfolge von echter und gefühlter Bedrohung, Verwundbarkeit, Gegenmaßnahmen sowie das Einspielen von Patches und Updates oder auch das Einstellen der Unterstützung eines Patch-Managements für Geräte mit älteren Betriebssystemversionen verspricht Einnahmen. Und dies für jede Seite. Denn IT-Sicherheit hat sich längst vom Geschäftsmodell zum *Big Business* entwickelt.

Erwerb von Wissenskurven – Teil 1
Ein weiterer nicht zu unterschätzender Aspekt ist der Zeitrahmen des Erwerbs von Fachwissen. So benötigt ein mittelbegabter, mittelfleißiger Mensch zum Aufbau eines wirklichen idealerweise allumfassenden Wissens mit einem soliden Wissensstand zur Umfänglichkeit digitaler Geräte und deren Kommunikationsmöglichkeiten nicht Jahre, sondern eher Dekaden. Die schiere Dauer dieses Wissenserwerbs in Bezug zu den Lebenszyklen heutiger Digitaltechnologie und Protokollvarianten offenbart jedoch schon,

dass es immer weniger ausgewiesene Experten geben dürfte, die tatsächlich jede Komponente dieser Geräte, alle algorithmischen, protokollspezifischen und kryptografischen Feinheiten, die Einzelheiten des gerade in diesem Gerät verbauten Protokoll-Stacks oder gar deren Zusammenwirken in seiner Gesamtheit wirklich verstanden haben.

Erwerb von Wissenskurven – Teil 2
Wir erkennen nicht zuletzt auch an den im Rahmen dieses Buches geschilderten Verwundbarkeiten, dass heutige Angreifer, die tatsächlich selbstständig Sicherheitslücken in mobilen Systemen aufspüren und diese nicht nur anwenden, über ein zum Teil sehr beachtliches IT-Wissen verfügen, das aus der neutralen Beobachtersicht einfach faszinierend ist. Allerdings verfügen Angreifer hinsichtlich des Erwerbs ihrer ‚Wissenskurven' gegenüber den Sicherheitsarchitekten, die ein Gesamtsystem schützen müssen, über den immensen Vorteil, dass sie sich dediziert mit einzelnen Aspekten auseinandersetzen können und es oftmals für einen erfolgreichen Angriff nicht erforderlich ist, die Gesamtarchitektur im Detail durchdrungen zu haben. Genau solch ein solides Gesamtverständnis ist aber für das Härten eines Gesamtsystems zwingend erforderlich. Diese Asymmetrie der erforderlichen Wissenskurven von Angreifern und Sicherheitsarchitekten wird ein ständiges Dilemma bleiben.

Lebenszyklus des Patch-Managements
Oftmals offenbart die tagtägliche Praxis in Betrieben und Behörden, dass der IT-Administrator und der IT-Security-Administrator beim Patch-Management einem Interessenkonflikt unterliegen. Während der IT-Administrator durch das Einspielen von Patches Kompatibilitätsprobleme befürchtet die ein stabil laufendes System stören könnten, befürchtet der IT-Sicherheits-Administrator in noch nicht eingespielten fehlenden Patches zu Recht ein Sicherheitsrisiko. Auch bei einem kontrollierten Patch-Management-Lebenszyklus vergeht so für gewöhnlich nochmals wertvolle weitere Zeit, bis ein bereits vorhandener Patch für eine bereits bekannte Schwachstelle in ein Produktivsystem eingespielt ist.

Vermutlich lässt sich diese Ursachenforschung und das Sammeln von Gründen noch beliebig weitertreiben. Nach meinem Verständnis ist es nicht zuletzt auch unter Einbeziehung der obigen allesamt nicht-technischen Gründe ab einer gewissen Komplexität digitaler, drahtlos kommunizierender Geräte definitiv nicht mehr möglich, diese gegen alle aktuell veröffentlichten, aktuell bekannten oder in naher Zukunft, innerhalb des Lebenszyklus des Gerätes, realisierbaren Angriffe zu härten. Heutige mobile Geräte haben diesen Grad an Komplexität schon längst überschritten. Ein weiterer nicht unerheblicher Grund ergibt sich unmittelbar aus dem Thema *‚Erwerb von Wissen':* ‚Low-hanging-fruits' lassen sich als *Pentester* oder *Fuzzer* oftmals schnell ernten und dies mehrheitlich verbunden mit einem sehr soliden Salär. Ist man dagegen innerhalb eines Unternehmens oder einer Behörde verantwortlich für die IT-Sicherheit, dann ist dieser Job verbunden mit einer enormen Verantwortung, Erreichbarkeit rund um die Uhr und einem teilweise gar nicht so üppigen Gehalt. Dafür aber erfordert dieser Job sehr viel Fachwissen. Abgekürzt könnte man aus diesem Betrachtungswinkel sagen: *„Offensive security beats defensive security."*

Ausblick mit Blick auf den Leser

14

Zum Ausklang dieses Buches fällt es mir schwer ein Resümee zu ziehen. Das erklärte Ziel war es dem Leser eine fundierte Einführung in die Chronologie bekannter Angriffe und Verwundbarkeiten auf mobile Systeme und dessen konzeptionelle Einordnung zu bieten. Der Leser sollte einen Überblick über die Vielfältigkeit nachweisbar ausgenutzter Angriffsvektoren auf verschiedenste Komponenten mobiler drahtloser Geräte sowie den teilweise inhärent sicherheitskritischen Aktivitäten moderner mobiler OS erhalten. Inwieweit dieser Überblick über die letzten zwei Dekaden mir tatsächlich gelungen ist können sie lieber Leser letztendlich nur selber entscheiden.

Als täglicher Nutzer diverser mobiler Geräte der mit diesem Buch einen ‚Blick unter die Motorhaube' gewagt hat sollten sie sich bewusster machen können was möglicherweise auf den mobilen Geräten im Hintergrund passiert die sie umgeben und wie angreifbar diese und sie dadurch sind. Vergegenwärtigen sie sich, in welchen Fällen sich durch ein möglicherweise korrumpiertes mobiles Gerät oder eine abgehörte Kommunikation handfeste Auswirkungen auf ihre realen Lebensumstände oder die ihrer Mitmenschen ergeben.

Als Studierende der Informatik, Wirtschaftsinformatik, Elektrotechnik oder verwandter Studiengänge: Selbst dann, wenn das Thema IT-Sicherheit nicht den Schwerpunkt innerhalb ihres Curriculums einnehmen sollte so ist es sicherlich von Vorteil ein gewisses Verständnis aufgebaut zu haben welche Schwachstellen und Verwundbarkeiten sich im Umfeld mobiler Geräte ergeben. Auch wenn einige der Themen in diesem Buch für ihre Gruppe der Leserschaft doch nur recht oberflächlich erörtert werden konnten, so können sie doch sicherlich als Einstieg für ein weiterführendes Studium der Materie dienen. Denn eines sollte offenkundig geworden sein: Spätestens als Entwickler von Anwendungen werden sie sehr schnell auch mit Sicherheitsfragen digitaler Technologie konfrontiert sein. Dann ist es vorteilhaft, wenn nicht sogar zwingend notwendig einordnen können wovon der Sicherheitsexperte in ihrem Entwicklerteam gerade spricht.

D. Westhoff, *Mobile Security,* https://doi.org/10.1007/978-3-662-60855-5_14

Als Praktiker, Entwickler, PenTester oder Reverser: Die IT hat schon lange ihre Unschuld verloren, wenn sie sie jemals hatte. Daher ist es gerade für Praktiker, Entwickler und PenTester erforderlich sich neben der unbestritten notwendigen Aneignung von technischem Rüstzeug auch eingehend mit ethischen und moralischen Fragestellungen zu beschäftigen. Eine Möglichkeit sich mit derartigen Fragestellungen zu Informatik und Ethik auseinanderzusetzen bietet beispielsweise die Seite ‚Gewissensbits' der Gesellschaft für Informatik (GI).

Als IT-Administratoren, IT-Sicherheitsbeauftragte oder Datenschutzbeauftragte: Der Stellenwert ihrer täglichen Arbeit kann nicht hoch genug eingeschätzt werden. Neben einer unabdingbaren Loyalität zum Arbeitgeber, sind Sie es die die digitale Gesamtarchitektur ihres Unternehmens sowie deren Abläufe in den Geschäftsprozessen im Detail durchdrungen haben sollten. Denn nur mit einem fundierten Gesamtverständnis kann das Betreiben und Härten des Gesamtsystems gelingen. Sie haben zu erörtern ob die Einbindung mobiler Geräte über Konzepte des Mobile Device Managements oder *Bring Your Own Device* (BYOD) vertretbar sind. In derartige Risikoabschätzungen sind auch die oben erörterten Asymmetrien der erforderlichen Wissenskurven von Angreifern und Sicherheitsarchitekten zu berücksichtigen. Gleichzeitig müssen sie inhaltliche Interessenskonflikte welche sich aus ihren unterschiedlichen Rollen ergeben kollegial, zielorientiert und professionell meistern.

Als industrieller Entscheidungsträger in KMUs, Konzernen oder Behörden oder auch als politischer Entscheidungsträger: Sie können nicht mehr sagen sie haben nichts gewusst. Gerade als Entscheidungsträger im industriellen Umfeld, aber auch in der Politik sollte ihnen bewusst sein, dass eine mehr und mehr um sich greifende Digitalisierung in den Arbeitsabläufen immer auch zu essentiellen Verwundbarkeiten von Geschäftsabläufen, Prozessen sowie einer Bedrohung der ‚Kronjuwelen' ihres Unternehmens führen kann. Eine Abwägung der digitalen Versprechungen gegen die im Rahmen dieses Buches aufgezeigten Bedrohungen ist daher dringend geboten. Und schließlich: Als politischer Entscheidungsträger werden sie nicht umhin kommen neben den Debatten um den Ausbau eines flächendeckend digitalisierten Wirtschaftsstandortes Deutschland auch die enormen gesellschaftlichen sowie geostrategischen Implikationen einer derartig feinmaschig vernetzten mobilen Kommunikationstechnologie zu begreifen und ungeschönt zu thematisieren. Gerade die Tatsache, dass weltweit zig Millionenfach eingesetzte mobile *Geräte des Consumer Bereichs* auch zu geostrategischen Zwecken nutzbar sind scheint in seiner gesamten Tragweite, wenn überhaupt, nur von einem kleinen Personenkreis ansatzweise erörtert zu werden. Zumindest sind alle bisherigen öffentlichen Debatten vorrangig geprägt von dem sicherlich ebenfalls zwingend notwendigen Diskurs inwieweit Komponenten einer 5G Infrastruktur von vertrauenswürdigen Netzausrüstern stammen sollten. Die mobilen Endgeräte selber scheint man im Zuge dieser Debatte völlig außen vor zu lassen.