Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Richtlinien für die Erstellung von CryptoExamples

Lisa-Marie Teis

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	Kai Mindermann M.Sc.
Beginn am:	2018-04-10
Beendet am:	2018-10-10

Kurzfassung

Die Kryptografie als Lehre der Verschlüsselung ist heute in der Zeit der Digitalisierung und somit auch in einer immer mehr technologiebasierten Gesellschaft unumgänglich und notwendig. Heute wird sie primär im Bereich der Informationssicherheit in Form der Anwendung von verschiedenen Konzepten und Algorithmen zur Verschlüsselung von Daten betrachtet. Um also unter anderem die Integrität und Vertraulichkeit von Daten gewährleisten zu können, ist es notwendig sich mit Kryptografie auseinanderzusetzen [Buc01].

In der folgenden Arbeit wurden Richtlinien für das Vorgehen zur Erstellung von Codebeispielen wichtiger kryptografischer Anwendungsfälle auf der Webplattform „CryptoExamples“[1] erarbeitet. Diese Richtlinien decken dabei den gesamten Prozess der Erstellung ab und machen dabei Vorgaben zu den Kriterien Minimalität, Sicherheit, Vollständigkeit, Kopierbarkeit, Dokumentation, automatische Testbarkeit und Anwendbarkeit. Dabei wurden sowohl sprachen- und anwendungsfall-unabhängige Richtlinien, aber auch speziell solche, die Vorgaben zu einem spezifischen Anwendungsfall oder einer Sprache machen, verfasst. Die erstellten Richtlinien wurden dann anhand konkreter Beispiele angewendet und eine weitere Evaluation der Richtlinien durchgeführt.

1 https://www.cryptoexamples.com/

Abstract

Cryptography, as the science of encryption, is indispensable and necessary today in the age of digitalization and thus in an increasingly technology-based society. Today, it is primarily considered in the area of information security in the form of applying various concepts and algorithms for data encryption. In order to be able to guarantee the integrity and confidentiality of data, it is necessary to deal with cryptography [Buc01].

In the following work, guidelines for the process of generating code examples of important cryptographic scenarios on the web platform 'CryptoExamples'[2] were established. These guidelines cover the entire process of creation and specify the criteria of minimality, security, completeness, copyability, documentation, automatic testability and applicability. Language- and scenario-independent guidelines, as well as concrete guidelines for a specific scenario or language have been written. The created guidelines were then applied on the basis of concrete examples and a further evaluation was carried out.

2 https://www.cryptoexamples.com/

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AE Authenticated Encryption

AEAD Authenticated Encryption with Associated Data

AES Advanced Encryption Standard

CBC Cipher Block Chaining

CCM Counter with Cipher Block Chaining (CBC)-MAC

Cipher Verschlüsselter Klartext

CMAC Cipher-based Message Authentication Code

DLIES Discrete Logarithm Integrated Encryption Scheme

ECIES Elliptic Curve Integrated Encryption Scheme

GCM Galois/Counter Mode

GMAC Galois Message Authentication Code

HMAC Hash Message Authentication Code

Integrität Sicherstellung, dass Daten nicht manipuliert werden

MAC Message Authentication Code

Modus Betriebsart, wie mit der Blochiffre verschlüsselt wird

PBKDF2 Password-Based Key Derivation Function 2

Plain Zu verschlüsselnder Klartext

RSA Rivest-Shamir-Adleman

SHA Secure Hash Algorithm

Tag Nachrichtenzusatz, der Integrität unterstützt

Vertraulichkeit Nur berechtigte Personen können Nachricht einsehen

1 Einleitung

1.1 Motivation

Viele Programmierer beschäftigen sich nur wenig mit Kryptografie und damit dem Generieren von sicherem Code. Oft fehlt ihnen das Fachwissen in diesem speziellen Bereich um sichere Software schreiben zu können. Aus diesen Gründen wird oft auf Programmbibliotheken für Kryptografie zurückgegriffen. Diese sind jedoch häufig nicht ausreichend dokumentiert um einfach essentielle Anwendungsfälle der Kryptografie umsetzen zu können. Es fehlt an konkreten Beispielen zum Verstehen der direkten Umsetzung, weswegen oft weitere Quellen herangezogen werden müssen um das gewünschte Ergebnis umsetzten zu können. Doch auch Beispiele, die im Internet gefunden werden, sind meist veraltet, nicht getestet, unsicher oder allgemein einfach schlecht dokumentiert. Das oft umständliche Suchen weiterer Quellen sowie unzureichende Dokumentation könnte durch gut beschriebene Codebeispiele für verschiedene Sprachen, Bibliotheken und Anwendungsfälle umgangen werden. Genau dies bietet die Plattform CryptoExamples. Sie stellt eine zentrale Sammlung von kryptografischen Codebeispielen für die wichtigsten Anwendungsfälle dar, die ebenfalls sicher, getestet und direkt ausführbar sind. Einige Beispiele in verschiedenen Sprachen und Bibliotheken sind zwar bereits vorhanden, jedoch sollen weitere folgen. Dafür fehlen der Plattform jedoch Richtlinien, die beschreiben wie genau ein vollständiges Beispiel erstellt werden muss. Auch zum Aktuell-halten und nachträglicher Überprüfung von Beispielen auf korrekte Bestandteile und Aufbau, ist das Vorhandensein von Richtlinien notwendig. Eben diese wurden im Laufe dieser Arbeit verfasst.

1.2 Ziel

Ziel der folgenden Arbeit ist die Erstellung von Richtlinien, die den gesamten Arbeitsprozess zum Erstellen eines vollständigen Gesamtbeispiels auf CryptoExamples umfassen. Dabei werden sowohl Richtlinien verfasst, die unabhängig von der genutzten Bibliothek, dem Anwendungsfall und der Sprache Aussagen machen, aber auch Richtlinien speziell für die genutzte Sprache (in diesem Kontext behandelt: Java, C# und Python) und die umzusetzenden Anwendungsfälle wie Hashing, digitale Signaturen, symmetrische und asymmetrische Verschlüsselung. Dabei sollen die erstellten Richtlinien wenigstens Vorgaben zu den Kriterien Minimalität, Sicherheit, Vollständigkeit, Kopierbarkeit, Dokumentation, automatischer Testbarkeit und Anwendbarkeit machen. Die Aussagen der Richtlinien werden also so verfasst, dass das entstehende Beispiel die genannten Kriterien möglichst gut umsetzt.

1.3 Beitrag

Es wurden Richtlinien zur Erstellung von Gesamtbeispielen für die wichtigsten kryptografischen Anwendungsfälle erstellt. Diese Richtlinien wurden in allgemeine, sprachen-abhängige und anwendungsfall-abhängige Richtlinien unterteilt, um die Richtlinien systematisch Evaluieren und Pflegen zu können, sowie den Erstellern der Codebeispiele die vollständige Umsetzung der jeweils zu beachtenden Richtlinien zu erleichtern. Die innerhalb der Arbeit verfassten Richtlinien wurden dem GitHub-Projekt CryptoExamples[1] innerhalb eines eigenen Repositorys[2] in englischer Form hinzugefügt, um so die Erhöhung des Bestands bereits vorhandener Codebeispiele bei gleichbleibender Qualität und Vereinheitlichung, das Aktuell-halten sowie die regelmäßige Überprüfung dieser zu unterstützen. Die Nutzung der Richtlinien wird ebenfalls durch eine erstellte Übersichtstabelle unterstützt, die die aktuell gültigen Sicherheits-Algorithmen und -Konzepte zusammenfasst. Die Tabelle enthält auch Angaben zur vermuteten Gültigkeitsdauer und einen Verweis auf herangezogene Quellen. Dies ermöglicht, dass nicht jeder Codeersteller selbst nach den aktuell sichersten Algorithmen und Konzepten suchen muss.

Neben den konkreten Richtlinien wurde eine Evaluation der Richtlinienpunkte anhand der Kriterien Minimalität, Sicherheit, Vollständigkeit, Kopierbarkeit, Dokumentation, automatische Testbarkeit und Anwendbarkeit durchgeführt, um so die Auswahl der Punkte zu begründen und darzulegen warum bestimmte Punkte zum späteren Vorhandensein der Kriterien im Gesamtbeispiel beitragen. Auch ein Lebenszyklus mit Zeitvorgaben sowohl für die Richtlinien selbst als auch für die Code-Beispiele wurde erstellt. Womit ebenfalls der zukünftig auf der Webseite CryptoExamples[3] durchzuführende Überprüfungs- und Instandhaltungsprozess genauer analysiert und beschrieben wurde. Des Weiteren erfolgte eine zusätzliche Evaluation durch die Anwendung der Richtlinien auf konkrete Beispiele der Webseite CryptoExamples sowie eine zusätzliche Evaluation mit Testpersonen, um so die Anwendbarkeit und Vollständigkeit der Richtlinien in der Praxis nachzuweisen.

1 https://github.com/cryptoexamples
2 https://github.com/cryptoexamples/CryptoExamples-Guidelines
3 https://www.cryptoexamples.com/

2 Verwandte Arbeiten

Es gibt unzählige andere Arbeiten, die sich genau damit auseinandersetzen und erklären wie spezifische Anwendungsfälle der Kryptografie funktionieren und dabei auch häufig auf deren Sicherheitsaspekte eingehen. Implementierungsdetails und deren Umsetzung werden aber meist nicht thematisiert und somit außen vorgelassen, wie ebenfalls in einer anderen Arbeit von Gutmann [Gut02] angesprochen wird.

Es existieren allgemein nur wenige vergleichbare Arbeiten, die speziell Richtlinien zur Erstellung von kryptografischen Codebeispielen oder allgemein Richtlinien für den Bereich der Kryptografie angeben. Eines der wenigen Paper von Peter Gutmann [Gut02] stellt Richtlinien für das Design von kryptografisch sicherem Code vor, mit deren Hilfe es möglich ist entstehende Schäden durch unerfahrene Nutzer im Bereich der Kryptografie zu vermeiden. Es werden in dieser Arbeit vor allem Beispiele aufgezeigt, in denen Kryptografie falsch angewendet wird. Dementsprechend werden ebenfalls mögliche Lösungen aufgezeigt.

Konkrete Richtlinien zur technischen Umsetzung für häufige kryptografische Anwendungsfälle finden sich aber unter anderem beim Bundesamt für Sicherheit in der Informationstechnik (BSI) [Bun18]. Hier wird erst einmal eine kurze Definition zum jeweiligen Anwendungsfall bzw. Konzept mit den wichtigsten Eigenschaften gegeben. Des Weiteren gibt es Verweise auf andere Webseiten, die erklären wie bestimmte Arten eines Schemas (z.B. Padding) oder Algorithmen (z.B. für Blockcipher) funktionieren. Darunter finden sich Anmerkungen, was bei den jeweiligen Schemata zu berücksichtigen ist. Hauptsächlich handelt es sich dabei um Anmerkungen bezüglich der Schlüssellänge oder anderer Sicherheitsaspekte. Eben solche Angaben werden auch in folgender Quelle von NIST (National Institute of Standards and Technology) [Bar16] gefunden. Ähnlich gibt es auch von dem „Open Web Application Security Project“ (OWASP) unter anderem zum Bereich Kryptografie (siehe Quelle [Thed]) eine Art Anleitung, in der Regeln angegeben werden, die eine sichere Umsetzung unterstützen. Es werden Algorithmen angegeben, die aktuell als sicher eingestuft sind, und was allgemein zu einem sicheren Vorgehen (wie z.B. die Schlüssel-Länge) gehört. Solche Anleitungen und Standards werden auch auf anderen Webseiten wie CERTStandard [CER] angeboten, auf welcher die Regeln sogar durch Codebeispiele erläutert werden. Allgemeiner werden auch Richtlinien für ein sicheres Design ([Thee]) oder konkrete Checklisten ([Thec], [Sea]) gegeben, an welchen man eine sichere Implementierung erkennen kann. Ebenfalls hilfreich um Implementierungen hinsichtlich ihrer sicheren Umsetzung zu überprüfen ist die nachfolgende Quelle von „Find Security Bugs“ ([Art]). Dort werden Code-Beispiele zu häufigen Umsetzungsfehlern gegeben, die den Code unsicher machen und eine Lösung dieses Fehlers wird ebenfalls durch ein Code-Beispiel angegeben.

Es gibt noch einige weitere Arbeiten wie von Whitten und Tygar [WT99], die sich zum Beispiel mit der Usability von Webseiten beschäftigen, bei denen Verschlüsselung eine

Rolle spielt. Dort wird die schlechte Handhabung und der daraus resultierende Unmut der Nutzer sich mit Verschlüsselung auseinander zu setzen aufgezeigt.
Ein anderes Paper von Anderson [And93] beschreibt warum kryptografische Systeme so oft versagen. Zwar bezieht sich seine Arbeit hauptsächlich auf den Bank-Bereich, aber auch er weist in seiner Arbeit darauf hin, dass Programmierern häufig die Kenntnisse in diesem Bereich fehlen und deswegen auch zukünftig Sicherheitslücken entstehen werden.

Andere vergleichbare Webseiten, die demselben Zweck dienen wie CryptoExamples, aber sich nicht nur auf den Bereich der Kryptografie beschränken, lassen sich einige finden, die ebenfalls zu aufkommenden Problemen hilfreiche Codebeispiele aufzeigen. So sind in diesem Kontext Webseiten wie StackOverflow[1] und andere Themen-spezifischere Webseiten wie Server Fault oder LaTex StackExchange zu nennen. Diese kümmern sich um die Lösung von Problemen der Nutzer in einem bestimmten Bereich. Ebenfalls haben diese (wie auch CryptoExample) den Anspruch, dass die Antworten sowie die Ideen zur Lösung des Problems als Code bestimmte Kriterien erfüllen sollen. Der Code muss minimal, lesbar, verifizierbar und komplett sein. Die Minimalität soll es dem User einfacher machen, die dargestellten Antworten zu verstehen bzw. den selbst gemachten Fehler auch zu erkennen und schnell ausfindig machen zu können. Minimalität heißt in diesem Kontext aber nicht so kurzfassen wie möglich, sondern auch die Lesbarkeit sollte weiter gegeben sein, d.h. schlüssige/verständliche Bezeichner und Kommentare, was dem in von CryptoExamples gefordertem Kriterium (siehe Kapitel 3.2) der Dokumentation gleichkommt.
Bei der Vollständigkeit fließt hier auch der Anspruch auf Kopierbarkeit mit ein, denn es müssen alle Dateien vorhanden sein um das Problem zu beheben, sodass der Fragende auch die Möglichkeit hat die Antwort mit einbauen und ausprobieren zu können um eventuell weiter auftretende Fragen stellen zu können. Jedoch müssen die beigetragenen Codeausschnitte keine kompletten Beispiele sein (in Java z.B. mit Klasse und Packages), es kann sich auch nur auf Ausschnitte, die das Problem lösen sollen, beschränken. Also müssen die Antworten auf den angesprochenen Webseiten in dem Sinne kopierbar sein, dass sie in den eigenen Code mit eingebunden, aber nicht selbst lauffähig sein müssen, was einen Unterschied zum Kopierbarkeits-Kriterium von CryptoExamples darstellt. Verifizierbarkeit ist in diesem Kontext nicht mit der für CryptoExample geforderten Testbarkeit vergleichbar. Hier geht es darum die Frage möglichst genau zu beschreiben und mit entsprechenden Problemen am eigenen Code zu unterlegen, sodass andere Nutzer dieses Problem als vorhanden verifizieren können. Die Codeausschnitte an sich müssen nicht testbar sein.
Auf Webseiten wie StackOverflow sind also im Bereich Minimalität, Dokumentation und Vollständigkeit in Bezug auf den Code Parallelen zu sehen, auch wenn es nicht auf kryptografischen Code beschränkt ist. Es sind jedoch nicht alle Kriterien (siehe Kapitel 3.2), welche die Beispiele auf CryptoExamples erfüllen sollen, gewährleistet.

Im Kontext der oben angesprochenen Webseite StackOverflow gibt es ebenfalls eine weitere Arbeit von Nasehi et al. [NSMB12], die anhand der dort gegebenen Antworten analysiert hat, was eine gute Lösung darstellt oder eben nicht. Dazu wurden sowohl die Qualität des dargestellten Codebeispiels als auch die beigefügte Erklärung zum Code, die von den Nutzern als gut bewertet wurde, analysiert und dadurch die Attribute für ein ins-

1 https://stackoverflow.com/

gesamt hilfreiches Codebeispiel herausgefiltert. Diese Erkenntnisse können in der folgenden Arbeit beim Schreiben eigener Richtlinien berücksichtigt werden um das Verständnis und die allgemeine Dokumentationsfähigkeit dieser zu verbessern. Die Haupterkenntnis war hier, dass eine gute Antwort (zumindest auf Implementierungsfragen) zu einem präzisen/minimalen Code auch eine Erklärung (bestenfalls mit Beispiel) oder alternativ eine gute Dokumentation zum Code mittels Inline-Kommentaren enthält. Ebenfalls wird aufgezeigt, dass ein Hauptteil der Fragen in den Bereich „How-to-do“ fallen, was noch einmal die Notwendigkeit von konkreten Codebeispielen unterstreicht. Auch wird darauf hingewiesen, dass Lernen anhand von korrekten Beispielen meist effizienter sei, als von Grund auf selbst ohne irgendwelche Vorkenntnisse eine Lösung erarbeiten zu wollen.

Weitere Foren wie Java/Python Forum[2] bieten auf Code-/Theorie-basierte Fragen Antworten. Auch hier wird ähnlich wie bei StackOverflow häufig mit Codeausschnitten geantwortet und diese somit bereitgestellt. Es wird aber explizit darauf hingewiesen, dass keine kompletten Lösungen angeboten werden, wobei hier also die Code-technischen Antworten noch weniger Kriterien (die bei CryptoExamples Voraussetzung sind (siehe Kapitel 3.2)) erfüllen als es noch bei StackOverflow der Fall war. Die Codes hier bieten zwar eventuelle Lösungsansätze auch für kryptografische Probleme, jedoch erfüllen diese keine der für CryptoExamples geforderten Kriterien, noch handelt es sich hier um eine sprachen-übergreifende, sondern sprachbezogene Plattform.

Man findet keine Webseite, die einen kompletten Überblick über verschiedene Code-technische Umsetzung in verschiedenen Sprachen und Bibliotheken speziell im Bereich der Kryptografie anbietet. Zwar gibt es immer wieder vereinzelt Tutorials über die Nutzung von vereinzelten Bibliotheken oder die konkrete Umsetzung verschiedener Einzelthemen, aber keinen Gesamtüberblick, der ebenfalls immer mit sicheren, getesteten und direkt ausführbaren Codebeispielen ausgestattet wäre.

2 https://www.java-forum.org/
https://www.python-forum.de/

3 Grundlagen

Innerhalb dieses Kapitels werden die notwendigen Grundlagen, welche zur Erstellung der Richtlinien benötigt wurden, genauer erläutert. Dies umfasst zum einen die Hauptaspekte der beachteten kryptografischen Anwendungsfälle und außerdem eine Erläuterung der an die Richtlinien gestellten Kriterien.

3.1 Wichtige kryptografische Konzepte

Die Empfehlungen für aktuell sicher geltende Algorithmen und Konzepte sowie die gegebenen Definitionen und Erklärungen basieren (wenn nicht weiter angegeben) auf der Quelle [Bun18].

Hashing: Formal beschrieben ist eine Hashfunktion eine injektive Funktion, die einen Bit-String beliebiger Länge auf einen anderen Bit-String einer festgelegten Menge abbildet. Die häufigste Anwendung ist dabei die Speicherung von Passwörtern. Auch wird die Hashfunktion oft verwendet um Schlüssel abzuleiten und die Integrität von Daten zu bewahren. Sie besitzt dabei drei Eigenschaften:

- Kollisionsresitenz: Zwei unterschiedliche Strings werden nicht auf dasselbe Element abgebildet
- One-way property: Ist ein Hashwert mit einer festen Länge gegeben, so ist es nicht möglich den String, der auf diesen Hashwert abgebildet wurde, nachträglich zurück zu ermitteln
- 2nd preimage property: Erweiterung der one-way property, nur dass der Hashwert eine beliebige Länge besitzt

Besitzt eine Hashfunktion alle oben aufgezählten Eigenschaften gilt sie als „cryptographically strong“. Hashfunktionen, die aktuell als sicher gelten, sind Funktionen ab Secure Hash Algorithm (SHA)-2, da für SHA-1 schon Eingriffe gefunden wurden, die die aufgezählten Eigenschaften verletzen. Zur SHA-2 Familie gehören etwa SHA-256, SHA-384 oder SHA-512, wobei die angegebenen Zahlen jeweils die Schlüssellänge repräsentieren. Auch gibt es zu SHA-2 bereits als Alternative SHA-3.

Salt: Werden beispielsweise zu den Passwörtern nur die zugehörigen Hashwerte gespeichert kann es zu Dictionary Attacken kommen, bei denen der Angreifer Möglichkeiten besitzt aus dem Hash das zugehörige Passwort zu ermitteln. Um dies zu verhindern wird nicht nur das Passwort, sondern auch das Salt, welches vorne an das Passwort gehängt wird, zusammen gehasht (siehe [Ins18]). Auch in anderen Anwendungsfällen findet Salt Verwendung. Es

handelt sich also lediglich um eine beliebig und zufällig gewählte Folge von Zeichen, die an den Plaintext (zu verschlüsselnder Klartext) aus Sicherheitsgründen angehängt wird, wie in der Arbeit von Gauravaram [Gau12] erläutert wird.

Symmetrische Verschlüsselung: Diese Art der Verschlüsselung dient dazu die Vertraulichkeit (confidentiality) zu erhalten. Es darf unberechtigten Personen nicht möglich sein Zugriff/Einsicht auf den Inhalt der Nachricht zu bekommen. Es existiert ein öffentlicher Schlüssel, der sowohl vom Sender zur Verschlüsselung, als auch vom Empfänger zur Entschlüsselung genutzt wird (siehe [Ins18]). Eine mögliche Art der Schlüsselerzeugung ist dabei etwa Password-Based Key Derivation Function 2 (PBKDF2).

PBKDF2: Aus Passwörtern wird mit ihrer Hilfe ein Schlüssel abgeleitet, der z.B. in der symmetrischen Verschlüsselung Anwendung findet. Auf das angegebene Passwort, mit einem Salt, wird hier eine kryptografische Hashfunktion oder HMAC angewendet. Dabei wird diese mehrmals (festgelegte Anzahl an Iterationen) auf das entstehende Ergebnis angewendet um die Sicherheit zu erhöhen. Als Informationen benötigt sie das Passwort, Salt, gewünschte Schlüssellänge, Iterationsanzahl und die zu verwendende kryptografische Funktion, wie in einer Veröffentlichung von NIST [TBBC10] erläutert wird.

Blockcipher: Als Cipher bezeichnet man allgemein den verschlüsselten Klartext. Mittels Blockcipher kann ein zu verschlüsselnder Klartext (Plain) fester Länge verschlüsselt werden. Dabei können verschiedene Algorithmen für die Blockcipher verwendet werden. Empfohlen werden etwa der Advanced Encryption Standard (AES)-128, AES-192 oder AES-256. Dabei gilt eine Blockcipher als sicher, wenn der genutzte zugrundeliegende Algorithmus sicher ist. Soll ein Plain verschlüsselt werden, der die festgelegte Länge überschreitet, wird der Plain in Blöcke der gewählten Länge geteilt und anschließend diese einzelnen Blöcke verschlüsselt (siehe BSI [Bun18]). Um mehrere Blöcke hintereinander verschlüsseln zu können werden verschiedene Betriebsmodi notwendig.

Betriebsmodus: Eine Blockcipher kann nur einen Block fester Länge verschlüsseln. Mittels eines ausgewählten Modus (Betriebsart, wie die Blockciffre verschlüsselt werden soll), können Plains, die in ihrer eigenen Länge ein Vielfaches der festgelegten Blocklänge besitzen, durch Zerlegung in Blöcke verschlüsselt werden. Als aktuelle Modi werden Counter with CBC-MAC (CCM) oder Galois/Counter Mode (GCM) (siehe Quelle [NSS]) empfohlen. Laut der Webseite [Art] sollte darauf geachtet werden, dass die Modi die Eigenschaft Authenticated Encryption with Associated Data (AEAD) umsetzen. Das bedeutet es wird nicht nur Vertraulichkeit, sondern auch Integrität (Daten können von außen nicht modifiziert werden) sichergestellt. Liegt kein Vielfaches der Blocklänge vor muss auf eine Variante von Padding zurückgegriffen werden, wobei aufgepasst werden muss, da eine falsche Padding-Variante laut BSI [Bun18] schnell zu einer Sicherheitslücke führen kann.

MAC: Message Authentication Code (MAC) basiert auf einem symmetrischen Schema, um die Integrität zu bewahren. Es wird ein öffentlicher Schlüssel sowie ein Validierungs- und ein deterministischer Tag-Algorithmus benötigt. Der Tag-Algorithmus bekommt die Nachricht sowie den Schlüssel übergeben und ermittelt daraus einen Tag. Dieser Tag (stellt eine Art Prüfsumme dar) wird neben der eigentlichen Nachricht an den Empfänger geschickt, der mittels des Validierungs-Algorithmus (der die erhaltene Nachricht, den Tag und

den Schlüssel übergeben bekommt) überprüfen kann, ob die erhaltene Nachricht verändert wurde, indem der Tag zu der erhaltenen Nachricht errechnet und mit dem erhaltenen Tag verglichen wird (siehe [Ins18]). Momentan empfohlen werden Hash Message Authentication Code (HMAC), Cipher-based Message Authentication Code (CMAC) und Galois Message Authentication Code (GMAC), wenn die zugrundeliegenden Hashfunktionen und Blockciphern ebenfalls sicher sind.

Asymmetrische Verschlüsselung: Bei dieser Art der Verschlüsselung gibt es einen öffentlichen und einen zugehörigen privaten Schlüssel. Die Nachricht wird mittels des öffentlichen verschlüsselt und durch den privaten Schlüssel wieder entschlüsselt (siehe [Ins18]). Ohne den privaten Schlüssel darf es also nicht möglich sein den ursprünglichen Plaintext zurück zu erhalten. Es ist ein Mechanismus zur Erzeugung des benötigten Schlüsselpaars notwendig und eine Ver- und Entschlüsselungsfunktion. Beispiel für solche Algorithmen sind etwa Rivest-Shamir-Adleman (RSA), Elliptic Curve Integrated Encryption Scheme (ECIES) oder Discrete Logarithm Integrated Encryption Scheme (DLIES).

Digitale Signatur: Sie wird dazu genutzt um die Integrität von Nachrichten zu bewahren. Es werden hier ein privater und ein öffentlicher Schlüssel sowie ein Tag- und Validierungs-Algorithmus benötigt. Der Tag-Algorithmus bekommt die eigentliche Nachricht und den privaten Schlüssel übergeben und bildet daraus den Tag (Nachrichtenzusatz, der Integrität unterstützt). Der öffentliche Schlüssel wird zum Validieren des Tags benutzt. Versendet wird der Tag, sowie die Nachricht. Dem Validierungs-Algorithmus wird beides und zusätzlich der öffentliche Schlüssel übergeben, wie in der Quelle [Ins18] genauer erläutert wird.

3.2 Kriterien

Die Aussagen der Richtlinien sollten so getroffen und verfasst werden, dass die mittels der Richtlinien entstehenden Gesamtbeispiele die nun folgenden Kriterien bestmöglich umsetzen und beinhalten:

Minimalität: Hier ist die Anzahl der Zeilen innerhalb des eigentlichen Codes, die für das Umsetzen des jeweiligen kryptografischen Anwendungsfalls benötigt wird, gemeint. Somit werden keine unnötigen Funktionalitäten oder allzu umständliche Umsetzungen eingebaut. Minimalität heißt hier jedoch nicht nur möglichst wenig Zeilen Code zu benutzen, sondern es wird lediglich soweit minimalisiert, dass die Lesbarkeit und Verständlichkeit noch weitergegeben sind d.h. es muss für den Lesenden noch möglich sein, die einzelnen Bestandteile und damit wichtige Funktionalitäten verstehen zu können.

Sicherheit: Da es sich hier um die Erstellung von kryptografischen Codebeispielen handelt, müssen die je nach Anwendungsfall genutzten Algorithmen und Konzepte, nach dem aktuellen Standard als sicher gelten, da sich deren Nutzung ansonsten erübrigen würde. Die Richtlinien müssen also so gestaltet werden, dass es möglich ist ein Beispiel zu erzeugen, das als aktuell sicher gilt. Des Weiteren sollten die Richtlinien auch dazu beitragen bereits vorhandene Beispiele aktuell und damit dem aktuellen Sicherheitsstandard entsprechend halten zu können.

Vollständigkeit: Die Richtlinien müssen den gesamten Arbeitsprozess zur Erstellung des Beispiels für die Plattform CryptoExamples darstellen. Das bedeutet zum einen, dass der Code alle zur Umsetzung notwendigen Funktionalitäten entsprechend dem Anwendungsfall besitzt und zum anderen auch, dass die Dokumentation, die außerhalb und innerhalb des Codes zum Verständnis von Bedeutung ist, ebenfalls durch die Richtlinien abgedeckt und damit beschrieben werden müssen.

Kopierbarkeit: Das gegebene Codebeispiel muss für den Nutzer so aufgebaut sein, dass dieser den Code direkt kopieren, in seine Programmierumgebung einfügen und ohne Probleme ausführen kann. Dazu gehört ebenfalls innerhalb der Dokumentation darauf hinzuweisen, wenn spezielle Anforderungen erfüllt werden müssen (z.B. bezüglich der Bibliothek), bevor der Code problemlos von anderen benutzt werden kann.

Dokumentation: Das gesamte Beispiel ist ausreichend zu dokumentieren, sodass es gut verständlich, lesbar und nachvollziehbar ist. Dazu gehört die innere und die äußere Dokumentation. Die äußere Dokumentation umfasst Angaben außerhalb des eigentlichen Codes (z.B. Autor). Die innere Dokumentation beinhaltet Anmerkungen und Kommentare zur Nutzung innerhalb der eigentlichen Implementierung.

Testbarkeit: Die Implementierung sollte automatisch testbar sein, d.h. man muss die Korrektheit des umgesetzten Anwendungsfalles überprüfen können. Damit bezieht sich dieses Kriterium auf die automatische Testbarkeit durch Testwerkzeuge.

Anwendbarkeit: Im Gegensatz zu allen anderen Kriterien bezieht sich dieses Kriterium nicht auf den Inhalt, sondern auf die Richtlinien selbst, was dieses Kriterium von den anderen Kriterien unterscheidet. Alle im folgenden erstellten Richtlinien sollten einfach zu überprüfen sein. Das bedeutet, dass alle Bestandteile der Richtlinien einfach auf ihr Vorhandensein im fertigen Beispiel abgeprüft werden können.

3.2.1 Beziehung zwischen den Kriterien

Die meisten der Kriterien beeinflussen sich gegenseitig und führen teilweise zu Zielkonflikten, bei denen die Umsetzung des einen Kriteriums zum Ausschluss oder zumindest einer Einschränkung eines anderen führt.

Anwendbarkeit: Das Kriterium der Anwendbarkeit macht keine Aussage über die Eigenschaften der Inhalte der Richtlinien, sondern über die Überprüfbarkeit der gesamten Richtlinien, was sie von den anderen abhebt. Sie steht somit in keiner direkten Beziehung zu den anderen Kriterien, da Richtlinien auch überprüft werden können ohne das Aussagen über die anderen Kriterien enthalten sind.

Beziehungen:

Dokumentation und Sicherheit: Die fertigen Beispiele müssen über die Zeit auf dem aktuellen Stand der Informationssicherheit gehalten werden. Ist das gesamte Beispiel nicht ausreichend dokumentiert, etwa mit Verweisen auf offizielle Literatur (Webseiten, Tutorials usw.), dann kann sich die schlechte Dokumentation auch auf die Sicherheit auswirken.

Dokumentation und Kopierbarkeit: Ist die Dokumentation, etwa zur Einbindung und Installation notwendiger Bibliotheken, nicht oder nur unzureichend vorhanden, so können diese vom User nicht korrekt eingebunden werden. Dies führt dazu, dass das Beispiel für den User nicht mehr einfach kopier- und ausführbar ist.

Testbarkeit und Sicherheit: Ist die automatische Testbarkeit nicht gegeben hat dies eventuell Auswirkungen auf die Sicherheit. Ist das Beispiel nicht testbar, da etwa Teile der Implementierung fehlen, kann dies auch zum Fehlen von Anweisungen führen, was den Sicherheitsaspekt minimiert.

Vollständigkeit: Das Kriterium der Vollständigkeit kann sich auf alle weiteren Kriterien auswirken. Wird nicht der gesamte Arbeitsprozess abgedeckt, kann es sein, dass nicht alle Kriterien berücksichtigt und dementsprechend vernachlässigt werden. Wird etwa im gesamten Arbeitsprozess keine Aussage über die Erstellung einer Dokumentation gemacht (die Richtlinien, decken nicht den gesamten Arbeitsprozess ab), so leidet die Dokumentation darunter. Oder ist die Richtlinie im Hinblick auf die Implementierung nicht vollständig, wird eventuell nicht der gesamte kryptografische Anwendungsfall mit allen Funktionalitäten umgesetzt und ist nicht mehr für den User ausführbar hinsichtlich der eigentlich gewünschten Funktionalität, die hätte erfüllt werden müssen.

Zielkonflikte:

Minimalität und Dokumentation: Soll die Implementierung umfangreich dokumentiert werden, so wird der Minimalitätsfaktor verringert. Ist die Dokumentation jedoch zu gering, schadet dies der Lesbarkeit und dem allgemeinen Verständnis. Somit stehen beide im Zielkonflikt.

Minimalität und Testbarkeit: Aufgrund der Minimalität wird der Umfang der Implementierung möglichst gering gehalten, sodass für verschiedene Funktionalitäten keine extra Methoden verfasst werden, sondern sich die Umsetzung (aufgrund der Ausführbarkeit) meist auf eine Main-Methode beschränken lassen könnte. Jedoch schränkt dieses Vorgehen, die automatische Testbarkeit ein, da diese durch eine Implementierung mittels feinerer Gliederung (z.B. durch Methoden-Definition) besser umsetzbar wäre.

Minimalität und Vollständigkeit: Wird die Vollständigkeit der Richtlinien reduziert, so kann dies Einfluss auf die Minimalität haben. Umgekehrt kann die Minimalität durch Reduktion der Vollständigkeit erhöht werden.

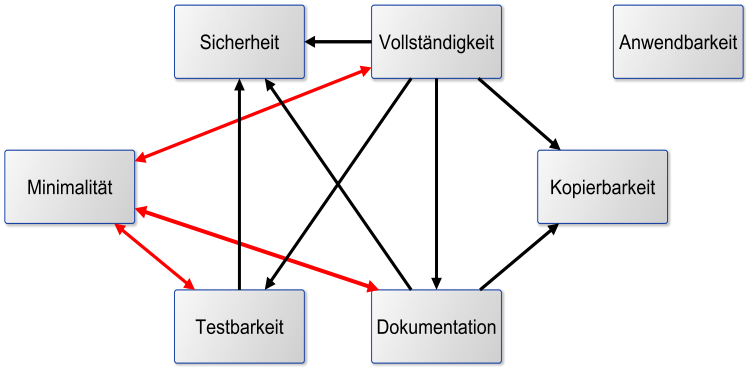

Abbildung 3.1: Beziehung und Zielkonflikte zwischen den geforderten Kriterien

3.2.2 Priorisierung

Da einige der Kriterien zu Zielkonflikten führen, müssen innerhalb der Erstellung der Richtlinien verschiedene Entwurfsentscheidungen, die entweder einen Fokus auf das eine oder andere Kriterium legen, getroffen werden. Aus den oben genannten Gründen ist eine Priorisierung der Kriterien vorzunehmen, die im Zielkonflikt stehen (Minimalität mit Testbarkeit, Dokumentation und Vollständigkeit), um verschiedene Entwurfsentscheidungen, die im Folgenden getroffen werden müssen, zu erleichtern.

So wird hier die Minimalität schwächer gewichtet, als die mit diesem Kriterium in Zielkonflikt stehenden anderen Kriterien.

Minimalität und Dokumentation: Ist der Code so minimal wie möglich gehalten, aber niemand kann mehr nachvollziehen welche Implementierungsentscheidungen getroffen wurden, macht dies den Code trotz minimaler Anzahl an Codezeilen unbrauchbar, da der User den Code und zukünftige Beteiligte, die etwa ein Review durchführen und/oder den Code aktualisieren wollen, den Code nicht mehr verstehen können. Dies würde sich also auch auf die Wartbarkeit des Codes auswirken. Demnach ist eine ausreichende Dokumentation speziell innerhalb der Implementierung von großerer Bedeutung.

Minimalität und Testbarkeit: Es soll möglich sein, die Funktionalität der Implementierung einfach testen zu können um überprüfen und ggf. auch beweisen zu können, dass sie die gewünschte Funktionalität besitzt. Ist der Code schlecht testbar, aber die Anzahl von Codezeilen gering, bringt diese Tatsache bezüglich des Wissens, ob die Implementierung überhaupt korrekt funktioniert, nichts. Es ist demnach wichtiger, die Funktionalität einfach überprüfen zu können als die Implementierung so kurz wie möglich zu halten.

Minimalität und Vollständigkeit: Decken die Richtlinien nicht den gesamten Arbeitsprozess zur Erstellung der Beispiele ab, so kann es später zu Problemen bei der Generierung dieser kommen oder sich auf die Qualität und Struktur dieser auswirken. Somit ist es wichtiger den gesamten Arbeitsprozess abzudecken, statt etwa den Umfang der Implementierung möglichst gering zu halten.

3.3 Vorgehen

Innerhalb der Arbeit wurden verschiedene Arten von Richtlinien erstellt. Diese lassen sich in verschiedene Ebenen gliedern und als Baum visualisieren (siehe Abbildung 3.2). Die Wurzel macht Aussagen darüber was sprachen- und anwendungsfall-unabhängig gelten muss. Die darunter folgenden Richtlinien sind zwar immer noch unabhängig vom kryptografischen Anwendungsfall, machen aber für alle hier betrachteten Sprachen (Java, C#, Python) Aussagen darüber was speziell bezüglich des Aufbaus in der jeweiligen beachtet werden muss. Dessen Kindknoten stellen die speziell für einen ausgewählten Anwendungsfall notwendigen Bestandteile, unabhängig von der genutzten Sprache, dar.

Bei der Erstellung der Richtlinien wurde jedoch „bottom-up“ vorgegangen. Zuerst wurde analysiert wie man einen ausgewählten Anwendungsfall in den behandelten Sprachen umsetzt und welche Bibliotheken dazu zur Verfügung stehen. Anhand von anfangs selbst erarbeiteten Beispielen zur Umsetzung des Anwendungsfalles und im Laufe der Arbeit veröffentlichten Beispielen auf der Webseite wurde die unterste Schicht der Richtlinien erstellt indem Ähnlichkeiten (in Beispielen mit demselben Anwendungsfall) innerhalb deren Struktur analysiert und zusammengefasst wurden.
Nachdem für alle hier betrachteten kryptografischen Anwendungsfälle (Digitale Signatur, Asymmetrische/Symmetrische Verschlüsselung, Hashing) die Richtlinien fertiggestellt waren, wurden für alle Beispiele, die in derselben Sprache verfasst wurden, Ähnlichkeiten in deren Aufbau gesucht. Aus den gefundenen Zusammenhängen entstanden die sprachenabhängigen Richtlinien der mittleren Ebene.
Für die Wurzel wurden die mittleren Richtlinien auf Gemeinsamkeiten analysiert. Gab es innerhalb dieser Gemeinsamkeiten, die sprachen-übergreifend auftraten, wurden diese Gemeinsamkeiten innerhalb der obersten Richtlinien zusammengefasst. Zusätzlich wurden Punkte eingefügt, die unabhängig von der eigentlichen Implementierung vorzunehmen sind.

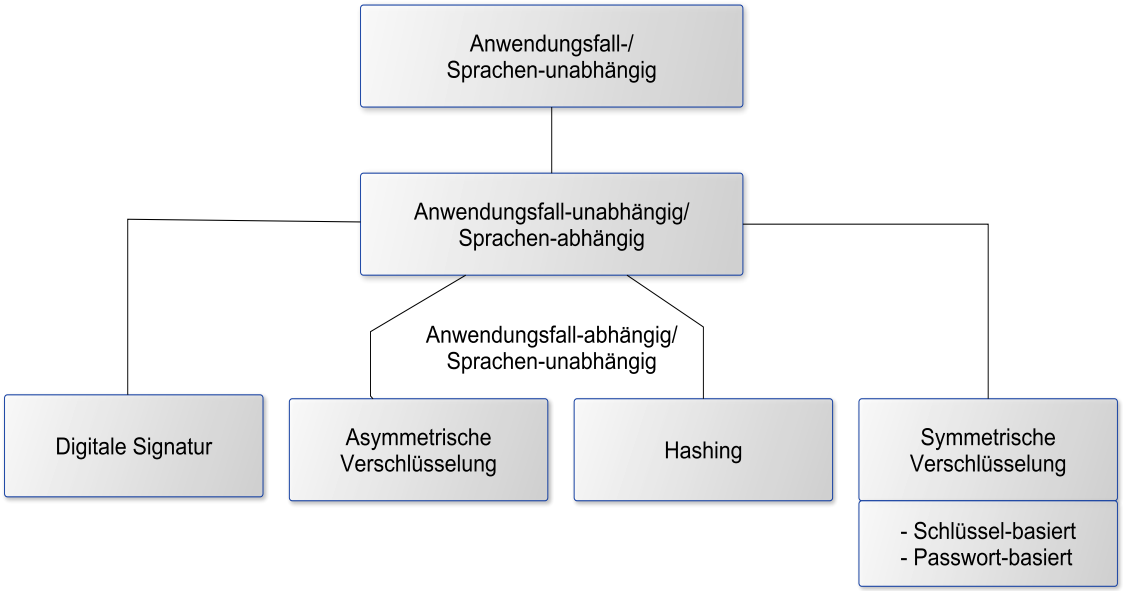

Abbildung 3.2: Visualisierung der Struktur der erstellten Richtlinien

3.4 Darstellung der Richtlinien

Innerhalb der Arbeit liegen die Richtlinien lediglich in textueller Form vor, jedoch werden die erstellten Richtlinien ebenfalls der Plattform CryptoExamples zur Verfügung gestellt. Ein Verweis auf die Richtlinien wird im Haupt-Repository[1] auf GitHub von CryptoExamples gegeben. Die Richtlinien werden in einem eigenen Repository[2] als Markdown-Datei (README.md) angegeben. Sie sind dort als Liste mit abhakbaren Checkboxen strukturiert. Damit wird es für den Benutzer einfach erkenntlich welche Punkte noch bearbeitet werden müssen und welche Punkte der Richtlinie bereits abgearbeitet wurden, was dem geforderten Kriterium der Anwendbarkeit entgegen kommt. Die README.md enthält dabei die erstellten allgemeinen, anwendungsfall-abhängigen und sprachen-abhängigen Richtlinien. Allgemein werden Punkte, die nicht nur manuell abgeprüft werden können, hervorgehoben und auf mögliche statische Codeanalyse-Tools explizit hingewiesen. In diesem Zusammenhang wurden die Richtlinien ebenfalls auf Englisch übersetzt um eine größere Anzahl an Menschen erreichen zu können. Außerdem wurde die Formulierung im Vergleich zur hier vorhandenen ausführlichen Version in der eigentlichen Darstellung für GitHub möglichst minimiert, indem die für die Bachelor-Arbeit strukturgebenden Zwischenüberschriften ausgelassen wurden.

1 https://github.com/cryptoexamples/CryptoExamples
2 https://github.com/cryptoexamples/CryptoExamples-Guidelines

4 Allgemeine Richtlinien

In diesem Kapitel werden Richtlinien angegeben, die unabhängig von Sprachen und kryptografischen Anwendungsfällen gelten. Ebenso wird ein Vorgehen zur Ermittlung sicherer Algorithmen und Konzepte vorgestellt, sowie der mögliche zukünftige Überprüfungsprozess der Richtlinien und erstellten Beispiele beschrieben, da diese Bereiche ebenfalls unabhängig von Sprachen und kryptografischen Anwendungsfällen zu betrachten und zu analysieren sind.

Bei den anwendungsfall-/sprachen-unabhängigen Richtlinien wird zwischen einer **äußeren** und einer **inneren Dokumentation** unterschieden. Die innere Dokumentation umfasst dabei die Dokumentation innerhalb des Beispiel-Codes. Die äußere Dokumentation beschreibt alle nötigen Dokumentationen, die außerhalb der eigentlichen Implementierung notwendig sind.

Der Aufbau eines gesamten Beispiels eines Codes mit vollständiger Dokumentation inklusive der konkreten Implementierung gestaltet sich wie folgt:

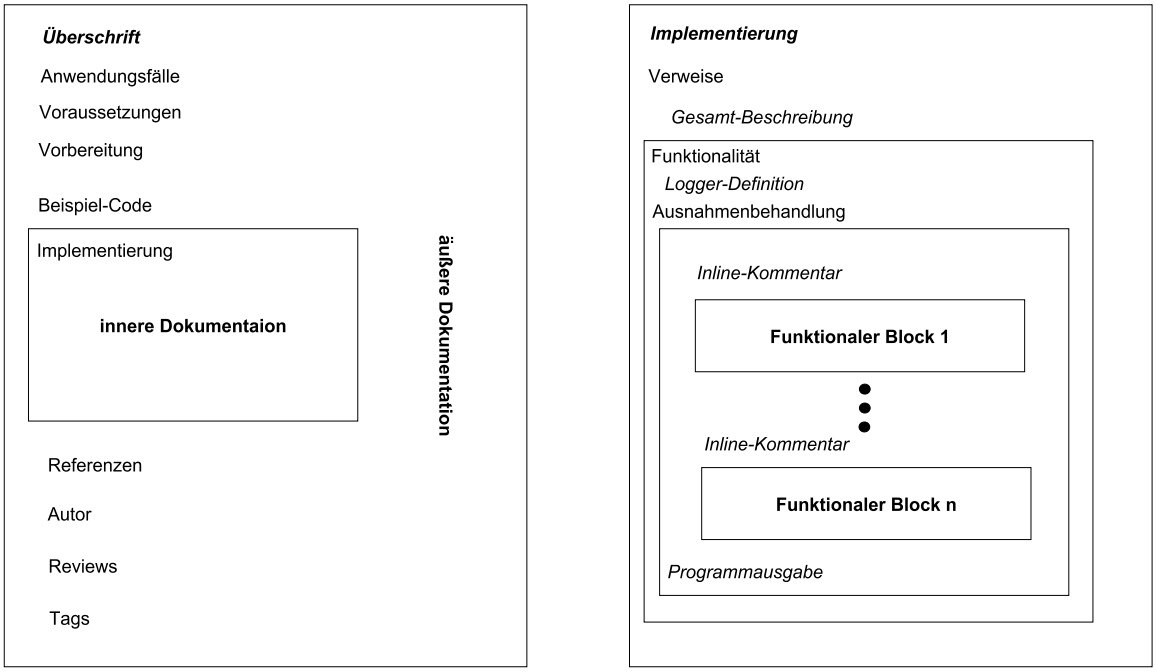

Abbildung 4.1: Allgemeiner Aufbau eines Gesamtbeispiels nach den Richtlinien

4.1 Äußere Dokumentation

Die äußere Dokumentation (wie in Abbildung 4.1 dargestellt) gestaltet sich wie folgt:

1. Gesamtbeispiel Überschrift
   a) Aufbau: $Programmiersprache$Anwendungsfall-Name using $Bibliothek
2. Anwendungsfälle
   a) Aufzählung der Anwendungsfälle für das folgende Codebeispiel
3. Voraussetzungen
   a) Verwendete Sprachversion
   b) Beschreibung zusätzlicher Anforderungen, damit das Beispiel funktioniert (z.B. Programmierumgebung, Betriebssystem)
4. Vorbereitung
   a) Werden neben den Standard-Bibliotheken andere genutzt, so muss beschrieben werden wie diese installiert und konfiguriert werden müssen
5. Beispiel-Code Überschrift
   a) Aufbau: Example Code for $Programmiersprache$Anwendungsfall-Name using $genutzte-Algorithmen-und-Konzepte
6. Referenzen
   a) Angaben zu entsprechenden offiziellen Dokumentationen, die Algorithmen, Konzepte, Parameter, Module, Pakete, Methoden oder Klassen genauer beschreiben
7. Autor
   a) $Vorname $Nachname
   b) Verweis auf das GitHub-Konto/-Profil
8. Reviews
   a) Datum des letzten Reviews: $YYYY-MM-DD
   b) $Vorname $Nachname desjenigen, der den Review durchgeführt hat
   c) Verweis auf das GitHub-Konto/-Profil
   d) $Versionsnummer der Richtlinien
9. Tags
   a) Genutzte Sprache, Bibliotheken, wichtige Konzepte und Verfahren innerhalb der Implementierung

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
1)-9)	Die vorhandenen Punkte stellen die minimale Anzahl notwendiger Angaben dar. Etwas wegzulassen würde die Gesamtqualität der Beispiele verringern

- Sicherheit:

6)	Durch gegebene Referenzen wird ermöglicht, sich über eingesetzte Elemente auch nachträglich informieren zu können. Durch Verweise auf offizielle Quellen soll nachträgliches Nachschlagen gewährleistet werden, sodass die Code-Beispiele, wenn notwendig (durch das Aufrufen der Referenzen) leichter angepasst und dadurch nachträglich auch einfacher verstanden oder aktualisiert werden können. Vom unterstützten Aktualisierungsvorgang profitiert auch wieder das Sicherheitskriterium
8)	Ein Code-Review dient generell auch dem Sicherheitsaspekt, da dadurch auch mögliche Sicherheitslücken gefunden werden können. Das Auflisten der letzten Review-Daten macht also für den User auch Aussagen über den Sicherheitsstatus des Beispiels

- Vollständigkeit:

1)-9)	Die Punkte stellen die gesamte äußere Dokumentation, die zur Erstellung eines CryptoExamples notwendig wird, dar

- Kopierbarkeit:

3)	Damit ein Code-Beispiel einfach kopierbar ist muss dokumentiert werden, wie das Beispiel erstellt wurde. Die Beispiele sollen stets in der neuesten stabilen Version der Programmiersprache oder des Compilers sein, jedoch muss der Nutzer die Möglichkeit besitzen, sehen zu können welche Voraussetzungen er erfüllen muss (bzw. welche Sprachversion momentan überhaupt aktuell ist), um abgleichen zu können ob er die Voraussetzungen überhaupt erfüllt
4)	Werden neben Standard-Bibliotheken weitere notwendig so muss dokumentiert werden, welche und wie mit diesen umgegangen werden muss, da das Beispiel ohne richtige Einbindung auf Nutzer-Seite nicht mehr einfach kopierbar ist

- Dokumentation:

1)-9)	Die in der äußeren Dokumentation zu findenden Punkte dienen alle der Dokumentation und damit der Übersichtlichkeit und dem Verständnis. Sie helfen den gegebenen Beispiel-Code zu verstehen und in den richtigen Fällen anzuwenden
1), 5)	Durch die gewählte Namenskonvention hat jedes Beispiel einen individuellen Namen, dem er einfach zugeordnet werden kann. Ebenfalls enthält er alle wichtigen Merkmale des Beispiels, sodass einfach anhand des Namens bereits Details zum Beispiel erkannt werden können und so das Auffinden des richtigen gewünschten Beispiels möglich wird
2)	Der spätere User der Website kann auf einen Blick erkennen, ob das folgende Beispiel seinen eigenen Anforderungen, die er umsetzen möchte, entspricht und kann so schneller selektieren, ob es die gesuchte Funktionalität umsetzen kann
3)	Dokumentation, die vor allem dazu nötig ist die Kopierbarkeit zu unterstützen
4)	Werden neben den Standard-Bibliotheken andere verwendet, ist es sinnvoll zu dokumentieren, wie mit diesen umzugehen ist. Ebenso, wenn keine Standard-Bibliotheken genutzt und andere verwendet werden müssen. Dieser Vorgang muss zumindest in den Grundzügen beschrieben (a)) und für den Fall auftretender Probleme auf weitere Literatur verwiesen werden. In diesem Kontext sind offizielle Webseiten zu empfehlen, da diese erstens vertrauenswürdiger sind und diese des Weiteren auch regelmäßig aktualisiert werden. Da auch derjenige, der den Beispielcode kopieren will, diese Schritte durchführen muss, kann es bei fehlender Dokumentation zu Problemen kommen. Ist dieser Vorgang nicht gut dokumentiert ist auch eine unkomplizierte Kopierbarkeit nicht mehr gegeben
6)	Die Referenzen enthalten alle Informationen, die dem Verständnis dienen und die Umsetzung genauer erläutern. Auch können diese Referenzen später hinzugezogen werden, wenn es darum geht Code-Beispiele aktuell zu halten, weswegen sie auch eine Art Sicherheitsaspekt erfüllen. Auch hier muss es sich um offizielle Quellen handeln, da diese erstens vertrauenswürdiger sind und diese des Weiteren auch regelmäßig aktualisiert werden
7)	Gibt eine Zuordnung des Beispiels zum Ersteller, auf den bei eventuellen Fragen verwiesen werden kann
8)	Zeigt auf wann und durch welche Person das letzte Code-Review durchgeführt wurde
9)	Durch die Angabe der Tags wird dem Nutzer ermöglicht ähnliche Beispiele für den ausgewählten Tag aufgezeigt zu bekommen

- Testbarkeit:

1)-9)	Die äußere Dokumentation umfasst nur Dokumentation und keine konkrete Implementierung und kann nicht getestet werden

- Anwendbarkeit:

1)-9)	Alle Punkte der Richtlinie können nur manuell als abhakbare Variante überprüft werden. Es gibt keine Möglichkeit zum Einsatz statischer Codeanalyse-Tools
	Neun Punkte sind in ihrer Anzahl noch so gering gehalten, dass sie im Gegensatz zu einer höheren Punktanzahl, noch mit überschaubarem Aufwand auf ihr Vorhandensein überprüft werden können

4.2 Implementierung

Die Implementierung muss grundsätzlich den aktuellen Programmierrichtlinien z.B. bezüglich Zeilenlänge, Bezeichner usw. folgen. Die Beispiele werden in englischer Sprache implementiert und dokumentiert.

1. Umsetzung mit der neuesten stabilen Version der Programmiersprache oder des Compilers
2. Verwendung von aktuell sicheren Algorithmen und Konzepten (siehe 4.3)
3. Auswahl der genutzten Bibliothek
    a) Primäre Nutzung von Standard-Bibliotheken (werden andere/weitere benutzt, auf die Aktualität und Sicherheit dieser achten)
4. Verweise notwendiger Bibliotheken bzw. Import-/Using-Verweise
    a) Explizite Angabe aller Importe
5. Innere Dokumentation
    a) Gesamt-Beschreibung
        i. Funktionale Beschreibung und Auflistung verwendeter Algorithmen und Konzepte des folgenden Codes
    b) Methoden-Kommentar
        i. Beschreibung der Funktionalität, Parameter und Rückgabewert
    c) Inline-Kommentare
        i. Beschreibung essentieller Teile des Codes (für Nutzer ohne sicherheitstechnischen Hintergrund)
6. Nutzung einer Logging-Komponente
    a) Keine Nutzung ungefilterter Systemausgaben wie System.out, print oder echo
7. Ausnahmebehandlung
    a) Ausnahmen werden am Ende abgefangen (um den Rest des Codes nicht zu überladen)
    b) Alle Ausnahmen außer Runtime Exceptions werden abgefangen
    c) Nicht einfach alle (z.B. in Java Exception e) sondern nur notwendige spezifische Fehler abfangen
    d) Ausgabe einer Standard-Fehlermeldung, keine angepassten Fehlermeldungen je nach Fehlerart
    e) Ausnahmen werden durch die Logging-Komponente ausgegeben
    f) Vermeidung der Ausgabe des gesamten Stacktraces

8. Programmausgabe
    a) Der Code zeigt, wie seine Sicherheitsfunktionalität geprüft werden kann (z.B. durch Protokollierung einer Ausgabe durch die Logging-Komponente, die den Erfolg oder Misserfolg angibt)
9. Kodierung
    a) Byte-Arrays werden für die Ausgabe kodiert
    b) Strings werden mit UTF-8 kodiert

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
4)	Alle nötigen Importe explizit angeben, da auch neue Elemente zu Bibliotheken später hinzugefügt werden können, die dann den Import des Beispiel-Codes nicht eindeutig machen. Importe, die eine Sammlung von Bibliotheken umfassen, sollen also gemieden und stattdessen explizit angegeben werden
5)	Kommentare werden nur an notwendigen Stellen gezielt eingesetzt, in denen sie dem Verständnis dienen und empfohlen werden (siehe Webseite [Jav17])
6) a)	Zwar würde zur Ausgabe von beliebigen Informationen ein standardmäßiger „Print“-Befehl genügen, jedoch wird aus Sicherheitsgründen von Quellen wie [Sea] und [Theb] auf die Nutzung von Loggern hingewiesen. Zwar bedeutet dies eine notwendige Definition des Loggers und schadet dem Minimalitäts-Kriterium, jedoch wird der Sicherheitsaspekt immer am stärksten in Entscheidungen mitberücksichtigt, weswegen sich für dessen Nutzung entschieden wurde
7) a), b), c)	Die Ausnahmebehandlung umfasst stets nur einen Block, anstelle eines Blockes um jede Funktionalität, die einen Fehler werfen könnte. Auch werden nur wirklich notwendige Ausnahmen abgefangen und nicht einfach alle existierenden. Es werden nur solche abgefangen, welche die konkrete Ausführung betreffen (keine Runtime-Exception, die durch korrekte Implementierung vermieden werden könnten)

7) d)	In diesem Falle wird zugunsten des Minimalitätsfaktors auf individuelle Fehlermeldungen verzichtet. Unter anderem wird aber auch von OWASP [Thec] darauf hingewiesen, lediglich eine generische Fehlermeldung zu erzeugen. Individuelle Meldungen würden zwar dazu dienen die Fehlerquelle einfacher zu finden, jedoch würde diese Ergänzung die Anzahl der Codezeilen (je nach Anzahl der abgefangenen Fehlerarten) wesentlich erhöhen und ebenfalls dazu führen, dass der Code an Übersichtlichkeit verliert
8)	Die Programmausgabe vergrößert zwar die Anzahl von Codezeilen, jedoch kann zum einen dem Nutzer durch einfache Ausführung des Programms gezeigt werden, dass die Funktionalität, die er erwartet, gegeben ist, zum anderen wird dadurch auch sichergestellt, dass der gesamte Arbeitsprozess eines Anwendungsfalles umgesetzt wird

- Sicherheit:

2)	Durch diese Angabe wird sichergestellt, dass die Umsetzung in allen erstellten Beispielen aller Anwendungsfälle sicher ist. Dies ist bei der Erstellung kryptografischen Codes eine der wichtigsten Ansprüche. Wie aktuell sicher geltende Algorithmen gefunden werden können wird in Kapitel 4.3 näher erläutert
3)	Bei der Auswahl der Bibliotheken, die zur Umsetzung genutzt werden, ist es empfehlenswert erst zu versuchen auf Standard-Bibliotheken zu setzen. Zum einen ist hier die Wahrscheinlichkeit größer, dass sie aktuell und damit auch sicher gehalten werden, zum anderen ist es für den User eventuell einfacher, da er sich nicht erst mit einer zusätzlichen Installation/Einbindung beschäftigen muss. Zusatzbibliotheken sind oft nicht sicher und veralten mit der Zeit, da sie nicht regelmäßig aktualisiert werden. Wird also eine andere/zusätzliche notwendig muss darauf geachtet werden, wie sicher und aktuell diese momentan noch ist
6), 7) e)	OWASP [Thec] rät zur Verwendung von Loggern, um keine sensitiven Daten nach außen dringen zu lassen (siehe Quelle [Theb]). Somit wird die Ausgabe von Nachrichten mittels des Loggers der normalen Standard-Ausgabe mittels „Print“-Anweisungen vorgezogen
7 f)	Auch die Quelle [Art] weist daraufhin, dass die Ausgabe des Stacktraces dem potentiellen Angreifer Informationen über die Implementierung geben könnte. Dies muss verhindert und damit vermieden werden

- Vollständigkeit:

	Beide Richtlinien (für die äußere Dokumentation und die Implementierung) umfassen zusammen den gesamten Arbeitsprozess, der Vorgaben zu allen nötigen Angaben zur Erstellung eines CryptoExamples macht
8)	Durch die verlangte Programmausgabe, wird sichergestellt, dass der gesamte Arbeitsprozess des umgesetzten Anwendungsfalles implementiert wurde, da die Programmausgabe sonst nicht erfolgen könnte

- Kopierbarkeit:

4)	Da die Richtlinien darauf hinweisen alle notwendigen Verweise vorzunehmen, werden alle folgenden verwendeten Funktionen und Komponenten erkannt. Auch, wenn später neue Elemente in einer Bibliothek hinzukommen, wird so sichergestellt, dass die verwendeten Komponenten eindeutig identifizierbar bleiben
9) a)	Durch die Repräsentation von Byte-Arrays mittels einer aktuellen Kodierung wird sichergestellt, dass die anzuzeigenden Zeichen auch überall erkannt und auf verschiedenen Geräten einheitlich dargestellt werden. Die Kodierung muss so gewählt werden, dass sie nur aus Zeichen besteht, die sich in jedem lateinischen Zeichensatz wiederfinden, wie auf der Webseite [Fas] beschrieben wird.
9) b)	Es muss hier eine weit verbreitete Kodierung gewählt werden. Durch die Repräsentation des Strings mittels dieser Kodierung, soll die korrekte Wiedergabe auf verschiedenen Geräten gewährleistet werden

- Dokumentation:

5)	Die Implementierung muss unabhängig vom Minimalitätsfaktor so dokumentiert/kommentiert werden, dass die Implementierung und darin enthaltene Entscheidungen gut verständlich sind, da das Kriterium der Dokumentation stärker gewichtet wird als das der Minimalität (siehe 3.2.2)
5) a), b)	Mittels der Richtlinie wird darauf hingewiesen, die Funktion des nachfolgenden Codes zu beschreiben. Zur Code-Qualität gehört allgemein eine Beschreibung der Funktionalität des Codes. Eine ausführliche Beschreibung muss auf Klassen-Ebene vorhanden sein. Des Weiteren sollte über vorhandene Methoden ein beschreibender Kommentar verfasst werden. Die Verfassung solcher Kommentare wird auch durch die Quelle [Ora] unterstützt und beschrieben
5) c)	Informationen und Entscheidungen müssen mittels Inline-Kommentar gekennzeichnet werden. Die Inline-Kommentare sollen laut der Webseite [Jav17] auf wenige Worte beschränkt sein und bedeutende Entscheidungen innerhalb des Codes festhalten. Dabei wird für jeden Block, der eine neue Funktionalität repräsentiert (wie ver- und entschlüsseln), ein Inline-Kommentar mit den getroffenen Entscheidungen angegeben
6), 7) e), 8)	Durch die Verwendung eines Loggers können sowohl Fehlernachrichten als auch Testergebnisse angezeigt und eventuell festgehalten werden

- Testbarkeit:

1)-9)	Die Implementierung im Gesamtbeispiel macht eine spätere automatische Testbarkeit erst möglich.

- Anwendbarkeit:

	Um zu überprüfen ob innerhalb der Implementierung die allgemeinen Programmierrichtlinien eingehalten wurden, kann mittels statischer Codeanalyse-Tools wie „Checkstyle“ (Java), „StyleCop“ (C#) oder „Pylint“ (Python) erfolgen
2)	Um Sicherheitslücken zu erkennen, können ebenfalls statische Codeanalyse-Tools verwendet werden. Für Beispiele in Java können „Xanitizer“, „SpotBugs/FindSecBugs“ oder „OWASP-Dependency-Check“ verwendet werden. Wobei in Xanitizer ebenfalls FindSecBugs und OWASP-Dependency-Check ausgeführt werden können. OWASP-Dependency-Check kann ebenfalls für Code in C# eingesetzt werden. Für Python wird für diesen Check „Bandit“ bereitgestellt. Um ein entsprechendes Codeanalyse-Tool auszuwählen kann ebenfalls auf der Webseite von OWASP in [Thef] recherchiert werden. Jedoch wird auf dieser Webseite auch explizit darauf hingewiesen, dass es viele Fälle gibt, bei denen nicht alle Sicherheitslücken restlos gefunden werden konnten, weshalb auch der Einsatz solcher Tools keine vollständige Garantie auf sicheren Code gibt
1), 3)-9)	Alle restlichen Punkte der Richtlinie können lediglich manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools

Anmerkung zur Dokumentation der allgemeinen Richtlinien: Es ist ebenfalls abzuwägen, was den Referenzen (äußere Dokumentation 6)) und was der inneren Dokumentation (5)) zugeordnet wird. Hier enthalten die Referenzen alle wichtigen Verweise auf Quellen, die das Nachschlagen und Sicherstellen der Verwendung von aktuellen Algorithmen gewährleistet, sowie das weitere Nachschlagen bzgl. verwendeter Komponenten ermöglicht. Diese Anmerkungen hätten jedoch auch als Inline-Kommentare z.B. über den Funktionalitäten der inneren Dokumentation zugeordnet werden können. Darauf wurde hier jedoch verzichtet, da die langen Links innerhalb der Kommentare die Codequalität und die Übersichtlichkeit verringern würden. Des Weiteren wird durch die Auslagerung in die Referenzen die Minimalität unterstützt. Für die Nutzer sind die Referenzen nicht von so hoher Relevanz und auf eventuell gewünschte Zusatzinformation kann auch einfach und unkompliziert (durch die Referenzen außerhalb der Implementierung) zugegriffen werden. Jedoch besteht durch die Auslagerung kein direkter Bezug mehr zwischen Referenz und Codezeile, der jedoch wünschenswert wäre.

4.3 Ermittlung sicherer Algorithmen/Konzepte

Hier sind sowohl die Sicherheit von Algorithmen zur konkreten Umsetzung (wie RSA, SHA usw.), als auch Konzepte in Form von allgemein notwendigen Sicherheitsaspekten (wie z.B. Schlüssel- und Noncen-Länge (willkürlich gewählte Nummer, die nur einmal verwendet wird), Modi usw.) eingeschlossen. Bei der Erstellung von kryptografischem Code ist es zwingend notwendig sich darüber zu informieren, welche Algorithmen und Konzepte zur Umsetzung des gewählten Anwendungsfalles genutzt werden sollten. Es werden neue Algorithmen erstellt und für früher als sicher geltende Algorithmen werden mögliche erfolgreiche Angriffsstrategien gefunden, die deren Sicherheitsaspekt zerschlagen. Demnach ist es wichtig auf dem aktuellen Stand der Informationssicherheit zu bleiben und sich vor der Erstellung eines Beispiels zu informieren. Grundsätzlich ist es empfehlenswert sich bei der Auswahl auf offizielle, vertrauenswürdige Quellen zu stützen, da diese wissenschaftlich fundiert sind und aktuell gehalten werden. Anzusprechen wäre in diesem Kontext das National Institute of Standards and Technology (NIST)[1]. Es erstellt Empfehlungen für alle hier angesprochenen Anwendungsfälle und weitere. Es macht darin Aussagen, was gerade als Sicherheitsstandard gilt und wie kryptografische Anwendungsfälle momentan sicher umgesetzt werden können. Dazu gibt man einfach den gesuchten Anwendungsfall in die Suchleiste der offiziellen NIST-Webseite ein. Besonders hilfreich sind hier die Suchergebnisse, die sich auf eine „Recommendation“ beziehen, die also Empfehlungen zur Umsetzung des Anwendungsfalles darstellen. Diese Dateien können auch einfach und schnell gefunden werden, indem in das Suchfeld einer beliebigen Suchmaschine der gesuchte Anwendungsfall/Konzept mit dem Zusatz NIST angeben wird.
Ebenfalls passende Informationen bietet das Bundesamt für Sicherheit in der Informationstechnik (BSI)[2], auch dieses enthält Richtlinien für den momentanen Stand der Informationssicherheit wie etwa in folgender Quelle [Bun18].

Speziell wichtig ist immer auch die richtige Schlüsselwahl. Die Länge muss beachtet werden, da sie maßgeblich für die Sicherheit ist. Die möglichen Schlüssellängen werden unter anderem auch in NIST-Veröffentlichungen wie [Bar16] näher erläutert. Grundsätzlich gilt die Nutzung von längeren Schlüsseln meist als sicherer, jedoch muss ab einer gewissen Länge auch zwischen Sicherheit und Schnelligkeit abgewogen werden. Eine Empfehlung bezüglich der Schlüssellänge wird dementsprechend auch von „BlueKrypt“[3] angeboten. Natürlich sind die Längen, welche überhaupt gewählt werden können, grundsätzlich auch immer von den gewählten Algorithmen abhängig. Insgesamt soll bei der Auswahl der Sicherheitsmechanismen für das Gesamtbeispiel, das erstellt werden soll, wie folgt vorgegangen werden:

1. Konsultiere mindestens zwei vertrauenswürdige Quellen wie NIST oder BSI für die Auswahl eines bestimmten Algorithmus
2. Konsultiere mindestens zwei vertrauenswürdige Quellen wie BlueKrypt oder NIST für die Wahl der Schlüssellänge

1 https://www.nist.gov/
2 https://www.bsi.bund.de/DE/Home/home_node.html
3 https://www.keylength.com/en/compare/

Auf Grundlage der genannten Quellen besteht die Möglichkeit eine Übersichtstabelle mit aktuell sicheren Algorithmen und Konzepten zu erstellen und auf der Plattform zur Verfügung zu stellen. Diese könnte dann von den Beispielerstellern genutzt werden, um den gewählten Anwendungsfall mit sicheren Mechanismen zu implementieren. Die Sicherheitsmechanismen werden dabei mit entsprechenden Quellen und ggf. mit einem voraussichtlichen Rahmen ihrer Gültigkeitsdauer versehen. Die aktuelle Tabelle würde sich nach dem aktuellen Sicherheitsstandard wie folgt gliedern:

Konzept	Auswahlmöglichkeit	Quelle
Hashing	SHA-2: SHA-256/384/512 SHA-3: SHA3-256/384/512 Länge ≥ 256 Bit	[Bun18](2018-2022), [Bar16], [NSS], [Gir18](BSI, 2018-2022)
Symmetrische Verschlüsselung		
Blockcipher	AES-128/192/256 Blockgröße ≥ 128 Bit	[Bun18] (2018-2022), [Bar16], [NSS]
Modi	CCM, GCM (nur authentifizierte Verschlüsselung anwenden)	[Bun18] (2018-2022), [Dwo11], [NSS]
Schlüssel	≥ 128 Bit	[Gir18] (BSI, 2018-2022)
Salt/Nonce	≥ 128 Bit (abhängig vom gewählten Algorithmus)	[TBBC10]
MAC	CMAC, GMAC, HMAC Blockgröße ≥ 128 Bit Tag-Länge ≥ 96 Bit	[Bun18] (2018-2022), [Bar16]
Passwort-Ableitung		
PBKDF2	HMAC Iterationen ≥ 10.000	[TBBC10], [GFN+17]
Asymmetrische Verschlüsselung		
RSA	Schlüssellänge ≥ 2.000 Bit (2.048 bzw. 4.096 Bit)	[Bun18] (2018-2022), [BD15], [NSS]
ECIES	Schlüssellänge ≥ 250 Bit	[Gir18] (BSI, 2018-2022)
DLIES	Schlüssellänge ≥ 2.000 Bit	[Bun18] (2018-2022)

Tabelle 4.1: Übersicht über aktuell sicher geltende Algorithmen und Konzepte

Die Inhalte der Tabelle wurden dabei wie folgt ausgewählt:

1. Es wurden ausschließlich vertrauenswürdige Quellen für die Auswahl eines bestimmten Algorithmus herangezogen (von Webseiten wie NIST oder dem BSI)
2. Es wurden ausschließlich vertrauenswürdige Quellen für die Auswahl der Schlüssellänge herangezogen (von Webseiten wie BlueKrypt oder NIST)

Die zu den Sicherheitsmechanismen gegebenen Quellen sollten anders als hier in der Ausarbeitung, bei einer Veröffentlichung explizit darunter angegeben oder mit einem Hyperlink zur entsprechenden Quelle versehen werden. So wird ebenfalls eine eigene weitere Recherche ermöglicht. Ebenso sollten die Abkürzungen der Konzepte beim ersten Auftauchen erläutert werden.

Eine andere Möglichkeit wäre es auch lediglich die Quellen (zur eigenen Recherche) zur Verfügung zu stellen. Also muss hierbei abgewägt werden, ob eine solche Übersichtstabelle gegenüber der bloßen Angabe von Quellen vorzuziehen ist oder nicht. Im Folgenden werden einige Vor- und Nachteile zur Angabe einer solchen Tabelle gegeben:

Vorteile:

1. Durch das Bereitstellen der Tabelle wird der aktuelle sicherheitstechnische Stand der Codebeispiele mit einem Blick erkennbar.
2. Die Angabe einer Übersichtstabelle erleichtert für die Beispielersteller die Erfüllung des Punktes 2) der Richtlinie 4.2. Sie müssten sich nicht mehr selbst informieren sondern können mit weniger eigenem zeitlichem Aufwand einfach in der Übersicht nachschauen, welche Algorithmen und Konzepte verwendet werden sollen, ohne selbst recherchieren zu müssen.
3. Die Tabelle könnte genutzt werden, um neue Sicherheitsmechanismen grün und veraltete Sicherheitsmechanismen rot zu markieren. Dadurch würde dem Beispielersteller sofort signalisiert, dass entweder ein bestehendes Codebeispiel angepasst oder ein neues Codebeispiel erstellt werden kann. Gleichzeitig würde den Projektbeteiligten signalisiert werden, dass Code-Reviews sinnvoll wären. Die Tabelle würde als ein Steuerungselement für die Aktualisierung der Codebeispiele dienen.
4. Durch zusätzliche Angabe von Jahreszahlen für die Gültigkeit der Algorithmen und Parameter kann die Tabelle auch als Planungselement für die zukünftige Anpassung der Tabelle selbst und der Codebeispiele sowie der Reviews verwendet werden.
5. Für Neueinsteiger der Kryptografie bieten die Quellenangaben eine gute Möglichkeit sich in das Thema einzuarbeiten.
6. Die Quellenangabe in der Tabelle dient der Einschätzung der Zuverlässigkeit der Aussage über die Sicherheitsmechanismen. Und zukünftig kann über diese Quelle möglicherweise eine Kontrolle der Aussage erfolgen.

Nachteile:

1. Die Tabelle muss aktuell gehalten werden, was einen zusätzlichen zeitlichen Aufwand für Beteiligte der Webseite bedeuten würde. Es müssten regelmäßige Überarbeitungen und Nachkontrollen der Inhalte der Tabelle erfolgen (siehe Kapitel 4.4), um sie dem aktuellen Stand der Informationssicherheit anzupassen.

2. Die Tabelle enthält für die Projektbeteiligten keine Quelle, die Auskunft über neue Entwicklungen gibt. Werden zur Überprüfung des Sicherheitsaspekts lediglich die genannten Quellen besucht, wird man nicht auf neu erkannte Sicherheitslücken oder neu etablierte Algorithmen und Konzepte stoßen. Um solche Informationen zu erhalten hilft die Tabelle nicht weiter.
3. Anhand der Tabelle ist es nicht erkennbar für welche Programmiersprache der Sicherheitsmechanismus implementiert wurde. Zwar werden verschiedene Auswahlmöglichkeiten gegeben, jedoch ist nicht jede dieser Möglichkeiten auch in jeder Sprache implementierbar, was anhand der Tabelle erst einmal nicht erkennbar wird.

Die aufgeführten Punkte würden also für die Nutzung einer solchen Tabelle sprechen.

4.4 Überprüfungsprozess

Sowohl die Richtlinien als auch die veröffentlichen Beispiele müssen aktuell gehalten und somit überprüft werden. Der Überprüfungsprozess soll wie folgt ausgeübt und verfolgt werden:

- Überarbeitung der allgemeinen Richtlinien alle 2 Jahre

  Ein Review der allgemeinen Richtlinien kann innerhalb größerer Zeitintervalle getätigt werden, da sie Aussagen über die Art der äußeren Dokumentation oder der allgemeinen Kernpunkte der Implementierung machen. Sie beinhalten damit Punkte, die nur wenig Anpassung benötigen, da sie von außen nur wenig beeinflusst werden (etwa das Erscheinen neuer Sprachversionen oder kryptografischer Algorithmen). Die Punkte der äußeren Dokumentation folgen einer selbst gewählten Form, die nur angepasst werden müssen, wenn sich die Ansprüche geändert haben.

- Überarbeitung der sprachen-abhängigen Richtlinien einmal im Jahr (oder beim Erscheinen neuer Sprachversionen bzw. beim Ergänzen neuer Sprachen)

  Die sprachen-abhängigen Richtlinien müssen öfters auf ihre Gültigkeit überprüft werden als es etwa bei den allgemeinen Richtlinien der Fall ist, da deren Inhalte ebenfalls durch das Erscheinen neuer Sprachversionen (damit evtl. auch neuer sprachlicher Komponenten) beeinflusst werden. Verfolgt man etwa die zeitliche Abfolge der Veröffentlichungen neuer Sprachversionen von Python (siehe Webseite [Pyt]) oder Java (siehe [Wik]) lässt sich kein einheitliches Zeitintervall ermitteln. Meist erscheinen neue Versionen innerhalb von 1-2 Jahren, dementsprechend muss sich die Überarbeitung und damit auch Durchsicht dieser Richtlinien ebenfalls in diesem Zeitintervall bewegen.
  Ebenso ist das Überarbeiten der Richtlinien außerhalb des standardmäßigen zeitlichen Rahmens auch dann nötig, wenn das Hinzufügen einer Sprachen geplant ist.

- Überarbeitung der anwendungsfall-abhängigen Richtlinien alle 2 Jahre

  Ebenso wie die allgemeinen Richtlinien ist hier eine Überarbeitung nach einem längeren Zeitintervall ausreichend. Innerhalb dieser Richtlinien werden die Kernpunkte für die vollständige Implementierung des jeweiligen Anwendungsfalles angegeben. Die Anwendungsfälle, sowie deren Umsetzung und damit auch Funktionsweise, werden sich auch im Laufe der Zeit nur wenig oder zumindest nur minimal ändern.

- Überprüfung der Tabelle 4.1 der Sicherheitsmechanismen alle 2 Monate

  Zwar ist das tendenzielle Vorgehen für die Implementierung eines Anwendungsfalles meist gleich jedoch ändern sich die nach dem aktuellen Standard als sicher geltenden Algorithmen und Konzepte. Es werden neue Algorithmen entworfen und/oder es werden Sicherheitslücken in früher sicher geltenden Algorithmen und Konzepten gefunden. Da die dort enthaltene Übersichtstabelle jedoch nur sichere Algorithmen und Konzepte enthalten darf ist eine häufigere Durchsicht notwendig, um diese aktuell

zu halten. Es ist jedoch nicht voraussagbar wann etwa neue kryptografische Algorithmen entwickelt oder existierende als unsicher erklärt werden, weswegen das hier festgesetzte Zeitintervall nicht an speziellen Zeitschranken abgeschätzt werden kann. Aus diesem Grund ist es notwendig, diese Überprüfung innerhalb kürzerer Intervalle kontinuierlich durchzuführen, da das Aufrechterhalten des Sicherheitsaspektes der Webseite höchste Priorität hat.

- Sicherheits-Review existierender Beispiele zum Gewährleisten der Sicherheit alle 2 Monate (Gesamt-Review synchron zur Veröffentlichung veränderter Richtlinien)

  Damit bereits existierende Beispiele ausschließlich dem aktuellen Standard nach sichere Algorithmen und Konzepte einsetzen, muss der Review-Prozess zumindest bezüglich des Sicherheitsaspektes der Beispiele innerhalb kürzerer Zeitintervalle getätigt werden. Wie bereits in der Überprüfung der Tabelle 4.1 gibt es keine Zeitschranken, die als Grundlage für die Auswahl des Zeitintervalls dienen könnten. Da es jedoch Priorität hat, dass die Codebeispiele sicher bleiben, ist ebenfalls ein kürzeres Zeitintervall zu wählen. Alle 2 Monate müssen die Beispiele also zumindest auf bestehende Sicherheitslücken untersucht werden, da es nicht im Vorfeld voraussagbar ist wann neue kryptografische Sicherheitslücken entdeckt werden.
  Ein Review der Gesamtbeispiele muss durchgeführt werden, wenn eine überarbeitete Richtlinie veröffentlicht wird, damit die existierenden Beispiele bereits nach einem kurzen Zeitintervall wieder mit den Richtlinien konform sind.

- Zentrale Bekanntgabe bevorstehender Überprüfungsprozesse

  Damit bevorstehende Überprüfungsprozesse vor allem nach längeren Zeitintervallen nicht versäumt werden, müssen die nächsten anstehenden Termine veröffentlicht oder zumindest die Beteiligten daran erinnert werden.

4.4.1 Regelmäßige Aktualisierung der Richtlinien

Zwar werden die hier erstellten Richtlinien so gehalten, dass sie auch in der Zukunft ohne größere Änderungen noch Gültigkeit besitzen, jedoch werden sich Erweiterungen und Anpassungen nicht vermeiden lassen. Der Lebenszyklus der Richtlinien lässt sich dabei wie in Abbildung 4.2 visualisieren. Wird eine Richtlinie erstellt wird sie mit einer Versionsnummer und dem Datum versehen und wird anschließend veröffentlicht. Danach wird die aktuell veröffentlichte Richtlinie z.B. für Reviews oder neue Beispiel-Erstellungen (siehe 4.2) genutzt. Je nach Art der Richtlinien erfolgt in dem oben (siehe 4.4) festgesetzten Zeitintervall eine Überarbeitung der aktuell bestehenden Richtlinien. Nach der Überarbeitung werden diese von einem Projektbeteiligten mittels einer Begutachtung überprüft und ggf. zur Veröffentlichung freigegeben oder zu einer erneuten Überarbeitung gegeben. Fiel die Begutachtung positiv aus erfolgt die Veröffentlichung mit Angabe der neuen Versionsnummer und dem aktuellen Datum.

Da also auch in Zukunft sicherlich weitere Anpassungen an den erstellten Richtlinien vorgenommen werden müssen, ist es sinnvoll die Richtlinien zu versionieren. Dies ist

ebenfalls wichtig, um nachvollziehen zu können anhand welcher Richtlinien die Beispiele erstellt bzw. welche in den vorgesehenen Review-Prozessen genutzt wurden.

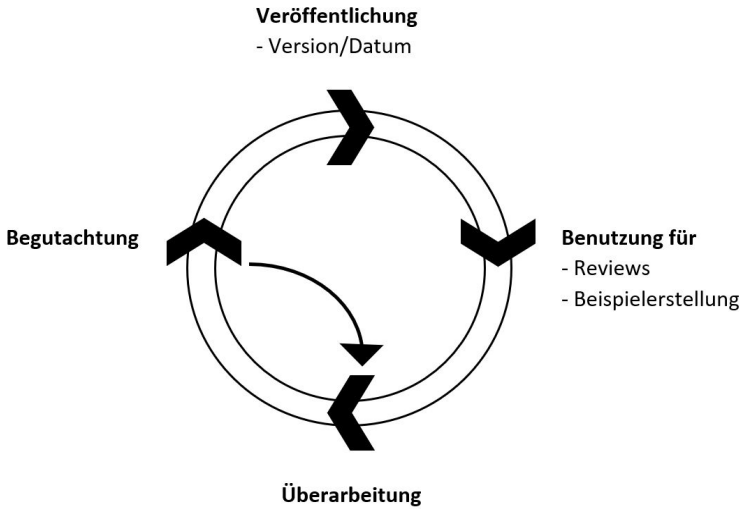

Abbildung 4.2: Lebenszyklus der Richtlinien

4.4.2 Regelmäßige Aktualisierung der Beispiele auf Basis der Richtlinien

Der Aktualisierungsprozess der Beispiele gliedert sich wie in den Abbildungen 4.3, 4.4 gezeigt. Die aktuelle Richtlinien-Version wurde veröffentlicht und ist danach in Benutzung (siehe Abbildung 4.2). Dabei kann sie zur Erstellung eines neuen oder zum Review eines bereits existierenden Beispiels genutzt werden. Wurde ein Beispiel neu erstellt folgt eine Begutachtung durch einen Projektbeteiligten und eine anschließende Veröffentlichung bzw. es wird zur Korrektur zurück zum Beispielersteller gegeben (siehe Abbildung 4.3).

Gibt es bereits existierende Beispiele so müssen diese auf dem aktuellen Stand gehalten werden. Um dies zu gewährleisten werden innerhalb von regelmäßigen Zeitabständen (siehe Kapitel 4.4) Reviews von Beteiligten durchgeführt. Die Beispiele werden dabei gegen die aktuell gültigen Richtlinien geprüft. Es folgt ggf. eine Überarbeitung der Beispiele, sowie eine Begutachtung durch einen Projektbeteiligten. Danach kann das überarbeitete Beispiel veröffentlicht oder zurück zu einer weiteren Überarbeitung gegeben werden (siehe Abbildung 4.4). Momentan findet noch kein regelmäßiger Review-Prozess statt. In Zukunft sollen sie jedoch in regelmäßigen Zeitabschnitten wie in Kapitel 4.4 beschrieben vorgenommen werden. Wurde ein Review durchgeführt wird das Datum und die Richtlinien-Version (anhand der überprüft wurde) vermerkt (siehe Richtlinie 4.1, Punkt 8)) und dies zeigt dem Nutzer so entsprechend wann das Beispiel zuletzt überprüft wurde bzw. wie aktuell das Beispiel ist.

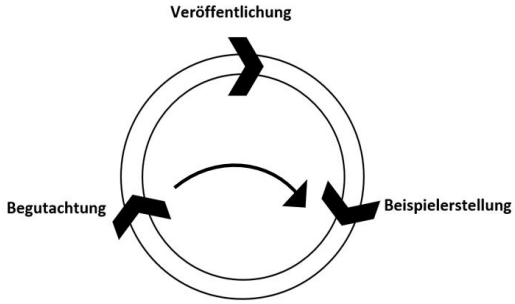

Abbildung 4.3: Vorgang bei der initialen Erstellung eines Beispiels

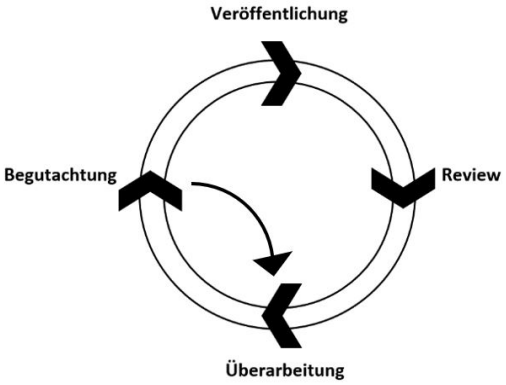

Abbildung 4.4: Lebenszyklus eines existierenden Beispiels

5 Sprachen-abhängige Richtlinien

Diese Richtlinien sind ebenfalls vom Anwendungsfall unabhängig, jedoch sprachen-abhängig und sollen somit Anleitungen geben, wie die jeweiligen Codebeispiele in den verschiedenen Sprachen aufgebaut werden sollten. Im Kontext dieser Arbeit wurden die Programmiersprachen Java, C# und Python betrachtet.

Die konkreten Aussagen der sprachen-abhängigen Richtlinien (Kapitel 5, 5.1) sind in Zusammenarbeit mit Nico Rusam im Kontext seiner noch laufenden Bachelorarbeit („Erstellung von CryptoExamples in C#“) entstanden ([Rus18]), um so den späteren einheitlichen und optimalen Aufbau (vor allem der C#-Beispiele) zu gewährleisten.

Die folgenden Punkte müssen in jeder der hier behandelten Sprachen (Java, C#, Python) vorhanden sein:

1. Definition Methode
    a) Name: demonstrate$Anwendungsfall-Name
    b) Definition eines entsprechenden Parameters
    (in der Regel ein String für den zu verschlüsselnden Plain)
    c) Rückgabe eines Ergebnisses abhängig von der Programmausgabe
    (siehe Richtlinien der Implementierung, Kapitel 4.2)
2. Definition Main-Methode
    a) Aufruf der Methode und Übergabe der Parameter
3. Beispiel besitzt ausschließlich die definierte und die Main-Methode

Für Implementierungen in Python müssen neben den oben genannten Punkten keine weiteren berücksichtigt werden.

Evaluation der Richtlinien anhand der Kriterien:

- **Minimalität und Testbarkeit:** Da durch die Richtlinien dieses Kapitels der grundsätzliche Aufbau der Implementierungen in der jeweiligen Sprache vorgegeben wird ergibt sich insbesondere hier ein Zielkonflikt zwischen Minimalität und Testbarkeit, wie bereits in Kapitel 3.2 angesprochen wurde. In diesem Falle muss zwischen den im Zielkonflikt stehenden Kriterien entschieden werden worauf der Fokus eher gelegt werden soll, was die Grundstruktur der Beispiele beeinflusst.

- Minimalität und Testbarkeit:

Nummer	Begründung
1)-3)	Für den grundsätzlichen strukturellen Aufbau werden die Definition einer Methode sowie einer Main-Methode gefordert. Zwar wäre es aus Minimalitätsgründen günstiger den gesamten Anwendungsfall innerhalb der Main-Methode einzubauen (auch zur eigentlichen Ausführbarkeit und damit Kopierbarkeit wäre dies ausreichend), jedoch würde dies die automatische Testbarkeit unnötig verkomplizieren. Es müsste die Logger-Nachricht ausgewertet und damit auch verschiedene Streams implementiert werden. Lagert man die Funktionalität jedoch in eine eigene Methode aus, wird der Code zwar minimal länger, da zur weiteren Ausführbarkeit der Aufruf innerhalb der Main-Methode notwendig wird, das Testen wird aber wesentlich vereinfacht. Ebenso trägt die Auswahl von entsprechenden Rückgabe-Typen und Rückgabewerten dazu bei, dass etwa bei einem J-Unit-Test die Methode nur noch aufgerufen und der Rückgabewert mit dem Soll-Resultat abgeglichen werden muss. Dementsprechend wird hier das Kriterium der Testbarkeit über das der Minimalität gestellt (siehe Kapitel 3.2.2)

- Minimalität:

1) b)	Um den Plaintext nicht extra innerhalb der Methode definieren zu müssen, wird dieser als Parameter übergeben. Was ebenfalls auch zur Code-Minimalität beiträgt

- Sicherheit
- Vollständigkeit:

1)-3)	Die Punkte machen Aussagen über alle notwendigen sprachlichen Komponenten, die in der Sprache Python beachtet werden müssen. Bei einer Implementierung in Python werden neben diesen Punkten keine weiteren speziellen Angaben gebraucht

- Kopierbarkeit
- Dokumentation
- Testbarkeit:

1)-3)	Der generelle Aufbau sieht die Definition einer Methode sowie einer Main-Methode vor. Diese Methode enthält einen Rückgabewert, der es bei einem automatischen Testverfahren ermöglichen soll lediglich die Methode aufzurufen und ihn mit einem Soll-Resultat zu vergleichen

- Anwendbarkeit:

1)-3)	Alle Punkte der Richtlinie müssen manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools

5.1 Java/C#

Bei der Nutzung von Java und C# müssen jedoch weitere zusätzliche Punkte der nun folgenden Richtlinie beachtet werden:

1. Package/Namespace
   a) Aufbau: com.cryptoexamples.$Sprache
2. Definition Klasse
   a) Definition als „public“
   b) Name: Example$Anwendungsfall-Name
3. Definition Logger
   a) Definition als „private static final“
4. Definition Methode
   a) Definition als „public static“
   b) Angabe des Typs des Rückgabewertes abhängig von der Programmausgabe

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
3)	Es soll lediglich einen definierten Standard-Logger geben, der für alle Ausgaben benutzt werden kann. Somit kann dieser als „static final“ gekennzeichnet werden, da der Logger, wenn er einmal definiert und zugewiesen ist, nicht mehr geändert werden soll und muss. Somit stellt er eine Art Konstante dar
4 a)	Wird die Methode als static definiert kann auf eine Instanziierung eines konkreten Objekts auf dem die Methode aufgerufen wird verzichtet werden. Was den Aufruf in der Main-Methode erleichtert und verkürzt

- Sicherheit
- Vollständigkeit:

1)	Der Aufbau entspricht der Ordnerstruktur des momentanen GitHub-Repositorys
1)-4)	Zusammen mit den Aussagen der vorherigen Richtlinie (siehe Kapitel 5) werden alle notwendigen Aussagen bezüglich des sprachlichen Aufbaus gegeben

- Kopierbarkeit
- Dokumentation

- Testbarkeit:

2) a), 4) a)	Wird die Klasse als public definiert ist sie auch außerhalb der eigenen definierten Umgebung für jeden sichtbar und damit zugreifbar. Somit können Tests (z.B. J-Unit) auch außerhalb des packages/namespaces definiert und auf den Inhalt der Klasse zugegriffen werden, was sinnvoll ist, da man die Tests nicht immer direkt bei der Klasse sondern außerhalb umgesetzt haben möchte. Ebenso gilt dies für Methoden

- Anwendbarkeit:

1)-4)	Alle Punkte der Richtlinie müssen manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools

6 Anwendungsfall-abhängige Richtlinien

In diesem Kapitel werden Richtlinien für die Erstellung von Codebeispielen vorgestellt, die das Vorgehen abhängig vom jeweiligen Anwendungsfall abdecken sollen.

6.1 Richtlinien für Hashing

1. Auswahl des Hashverfahrens
    a) Erzeugung einer Instanz, die die gewählte Hashfunktion ausführt
2. Generierung des Hashs
    a) Kodierung des Plaintexts beachten
3. Erstellung einer String-Repräsentation des Hashs

Abbildung 6.1: Visualisierung der Richtlinien für Hashing

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
1)-3)	Die Richtlinien stellen die minimale Anzahl an Schritten für die Umsetzung des Hashing-Anwendungsfalles dar

- Sicherheit:

1)	Bei der Auswahl sollte wie in Kapitel 4.3 beschrieben vorgegangen oder sich auf die erstellte Tabelle sicherer Algorithmen und Konzepte (siehe Tabelle 4.1) bezogen werden

- Vollständigkeit:

1)-3)	Um hier die gesamte Funktionalität des Anwendungsfalles zu gewährleisten muss lediglich der entsprechende Hash für den gegebenen Plain erzeugt werden

- Kopierbarkeit:

2) a)	Durch die Beachtung der Kodierung des Plaintexts, soll ein einheitliches Resultat (hier der Hashwert) auf verschiedenen Geräten gewährleistet werden
3)	Damit eine einheitliche Darstellung von Byte-Arrays gewährleistet werden kann, werden diese etwa mit Base64 kodiert und damit eine String-Repräsentation erstellt

- Dokumentation
- Testbarkeit:

3)	Eine der gewünschten Eigenschaften einer Hashfunktion ist es, dass es nicht möglich seien sollte von einem erzeugten Hash, diesen wieder zum Plaintext zurück konvertieren zu können. Demnach ist hier eine einfache automatische Testbarkeit nicht gegeben. Jedoch könnte auf Konsistenz geprüft werden, indem man denselben Plain hintereinander zweimal hasht und die beiden so ermittelten Hashwerte auf Gleichheit prüft oder den Plain einmal hasht, den Wert abspeichert und dann nochmals mit dem Rückgabewert der aufgerufenen Methode vergleicht (Konsistenzprüfung). Um im später erstellten Testfall einen Vergleichswert zu besitzen, wird vom erstellten Wert eine String-Repräsentation erstellt (ebenfalls auch für die geforderte Programmausgabe)

- Anwendbarkeit:

1)-3)	Ob innerhalb der Implementierung alle aufgeführten Schritte und damit die gesamte Funktionalität abgedeckt wurde, kann lediglich manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools. Es können nur Sicherheitslücken etwa in Form einer unsicheren Hashfunktion mittels statischer Analysetools gefunden werden (siehe OWASP [Thef])

6.2 Richtlinien für symmetrische Verschlüsselung

In diesem Abschnitt werden Richtlinien für die Erstellung von Codebeispielen vorgestellt, die den Anwendungsfall der symmetrischen Verschlüsselung abdecken sollen. Unterschieden wird dabei zwischen der Schlüssel- und der Passwort-basierten (Schlüssel wird aus einem Passwort abgeleitet) symmetrischen Verschlüsselung. Dabei enthalten die Schlüssel- und die Passwort-basierte Verschlüsslung überlappende Bestandteile und fallen teilweise zusammen.

Der gesamte Arbeitsprozess der symmetrischen Verschlüsselung kann wie folgt dargestellt werden:

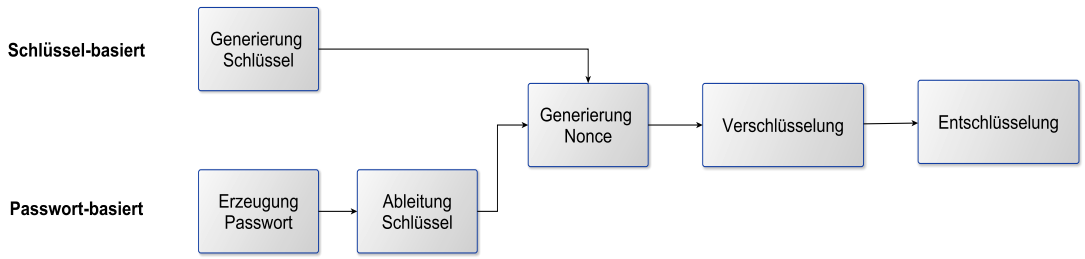

Abbildung 6.2: Visualisierung der Richtlinien für symmetrische Verschlüsselung

6.2.1 Schlüssel-basiert

1. Generierung Schlüssel
   a) Nutzung eines kryptografisch geeigneten Schlüssel-Generators bzw. Zufallszahlengenerators
2. Generierung Nonce
   a) Nutzung eines kryptografisch geeigneten Zufallszahlengenerators
3. Erzeugung einer Verschlüsselungsinstanz
   a) Auswahl des zu verwendenden Algorithmus, der Authenticated Encryption (AE) durchführt
4. Verschlüsselung des Plains
   a) Kodierung des Plaintexts beachten
   b) Zusätzliche Erstellung der String-Repräsentation der Cipher
5. Erzeugung einer Entschlüsselungsinstanz
6. Entschlüsselung der Cipher
   a) Zusätzliche Erstellung der String-Repräsentation der entschlüsselten Bytes

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
1)-6)	Die Richtlinien stellen die minimale Anzahl an Schritten für die Umsetzung der symmetrischen (schlüssel-basierten) Verschlüsselung dar

- Sicherheit:

1), 2)	Gibt es keine spezielle Art (z.B. Bibliothek-abhängig) den Schlüssel oder die Nonce zu generieren sollte man (wenn in der jeweiligen Sprache gegeben) einen als kryptografisch stark geltenden Zufallszahlen-Generator nutzen. Der genutzte Generator sollte einen gewissen Grad an Zufälligkeit besitzen um sichere/zufällige Zahlenfolgen zu generieren (siehe OWASP [Thea])
3) a)	Im Folgenden wird eine authentifizierte Verschlüsselung notwendig, um sowohl Vertraulichkeit als auch Integrität zu gewährleisten. Die bloße Auswahl der Blockcipher und des Modus ist also für die Generierung eines sicheren Beispiels der symmetrischen Verschlüsselung nicht allein ausreichend (laut Quelle [Art]). Es muss sich zumindest um einen authentifizierten Verschlüsselungsmodus handeln oder das Prinzip Encrypt-then-MAC angewendet werden um neben Vertraulichkeit auch Integrität zu garantieren
1), 2), 3), 5)	Zur Auswahl der Algorithmen und der Wahl der Schlüssellänge sollte sich auf einer der in Kapitel 4.3 genannten Quellen oder der erstellten Tabelle sicherer Algorithmen und Konzepte (siehe Tabelle 4.1) bezogen und informiert werden

- Vollständigkeit:

1)-6)	Um den gesamten Anwendungsfall abzudecken muss mit demselben Schlüssel ver- und wieder entschlüsselt werden können

- Kopierbarkeit:

4) a)	Durch die Beachtung der Kodierung des Plaintexts, soll ein einheitliches Resultat (hier der Cipher und der entschlüsselten Bytes) auf verschiedenen Geräten gewährleistet werden

- Dokumentation
- Testbarkeit:

6) a)	Um im später erstellten Testfall (sowie für die geforderte Programmausgabe) einen Vergleichswert zu besitzen, wird eine String-Repräsentation der entschlüsselten Bytes gefordert. So kann der gegebene Plain mit der String-Repräsentation der entschlüsselten Bytes auf Gleichheit überprüft werden, um so zu erkennen ob die Ver- und Entschlüsselung korrekt durchgeführt wurden

- Anwendbarkeit:

1), 2), 3), 5)	Auftretende Sicherheitslücken können mittels statischer Analyse-Tools entdeckt werden, so etwa ob ein kryptografisch geeigneter Zufallszahlengenerator oder die richtige Schlüssel-/Noncen-Länge verwendet wurde. Laut der Webseite [Art] kann damit auch entdeckt werden ob AE verwendet wurde
1)-6)	Das Vorhandensein aller einzelnen Bestandteile der Richtlinie können manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools

6.2.2 Passwort-basiert

Schritt 1) der vorherigen Richtlinie (siehe Kapitel 6.2.1) wird durch folgende Punkte ersetzt:

1. Erstellung eines Passwortes oder Annahme eines Passwortes für die Verschlüsselung
2. Ableitung des Schlüssels aus dem Passwort
   a) Ableitung des Schlüssels aus dem Passwort mittels eines sicheren Verfahrens
   b) Salt für PBKDF2 besitzt wenigstens die Größe, die für die genutzte Hashfunktion passend ist (z.B. 32 Bytes für SHA-256)

Danach folgen alle weiteren Schritte (ab 2)) aus 6.2.1.

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
	Die Richtlinien stellen die minimale Anzahl an Schritten für die Umsetzung der symmetrischen (passwort-basierten) Verschlüsselung dar
2)	Die Ableitung des Schlüssels vom Passwort ist aus Sicherheitsgründen laut NIST [TBBC10] zwingend notwendig und kann nicht ausgelassen werden

- Sicherheit:

1), 2)	Nur das Passwort dabei als Schlüssel zu benutzen sollte aus Sicherheitsgründen vermieden werden [TBBC10]
2)	Gibt es keine spezielle Art (z.B. Bibliothek-abhängig), die für die Schlüsselableitungsfunktion benötigten Größen (z.B. Salt) zu generieren, sollte man (wenn in der jeweiligen Sprache gegeben) einen als kryptografisch stark geltenden Zufallszahlen-Generator nutzen, da dieser laut OWASP [Thea] einen gewissen Grad an Zufälligkeit besitzen sollte um sichere/zufällige Zahlenfolgen zu generieren. Zur Auswahl von etwa benötigten Iterationszahlen oder Schlüssellängen sollte sich auf einer der in Kapitel 4.3 genannten Quellen bezogen oder sich in der erstellten Übersichtstabelle sicherer Algorithmen und Konzepte (siehe Tabelle 4.1) informiert werden

- Vollständigkeit:

1), 2)	Durch die zusätzliche Ableitung des Schlüssels aus dem Passwort, wird der gesamte Prozess der passwort-basierten symmetrischen Verschlüsselung umfasst

- Kopierbarkeit
- Dokumentation
- Testbarkeit
- Anwendbarkeit:

2)	Auftretende Sicherheitslücken können mittels statischer Analyse-Tools entdeckt werden, so etwa ob ein kryptografisch geeigneter Zufallszahlengererator, die richtige Schlüssellänge sowie eine sichere Iterationszahl oder kryptografische Funktion bei der Schlüsselableitungsfunktion verwendet wurden (siehe Quelle [Art])
1), 2)	Das Vorhandensein aller einzelnen Bestandteile, sowie die richtige Repräsentation der Variablen können aber nur manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools

6.3 Richtlinien für asymmetrische Verschlüsselung

1. Generierung eines Schlüsselpaares
2. Erzeugung einer Verschlüsselungsinstanz
   a) Nutzung des öffentlichen Schlüssels zur Verschlüsselung
3. Verschlüsselung des Plains
   a) Kodierung des Plaintexts beachten
   b) Zusätzliche Erstellung der String-Repräsentation der Cipher
4. Erzeugung einer Entschlüsselungsinstanz
   a) Nutzung des privaten Schlüssels zur Entschlüsselung
5. Entschlüsselung der Cipher
   a) Zusätzliche Erstellung der String-Repräsentation der entschlüsselten Bytes

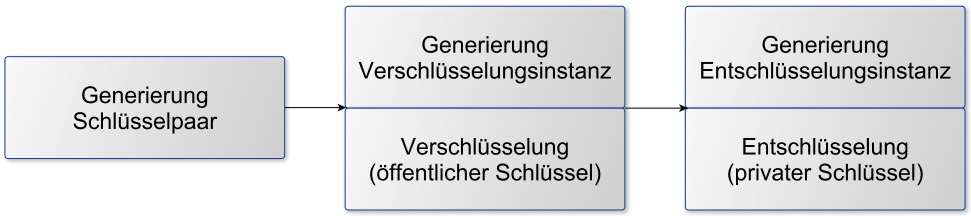

Abbildung 6.3: Visualisierung der Richtlinien für asymmetrische Verschlüsselung

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
1)-5)	Die Richtlinien stellen die minimale Anzahl an Schritten für die Umsetzung des Anwendungsfalls der asymmetrischen Verschlüsselung dar

- Sicherheit:

1), 2), 4)	Zur Auswahl der Algorithmen und der Wahl der Schlüssellänge sollte sich auf einer der in Kapitel 4.3 genannten Quellen bezogen oder sich in der Übersichtstabelle sicherer Algorithmen und Konzepte (siehe Tabelle 4.1) informiert werden

- Vollständigkeit:

1)-5)	Um den gesamten Anwendungsfall abzudecken muss mit dem öffentlichen Schlüssel ver- und mit dem zugehörigen privaten Schlüssel entschlüsselt werden

- Kopierbarkeit:

3) a)	Durch die Beachtung der Kodierung des Plaintexts, soll ein einheitliches Resultat (hier der Cipher und der entschlüsselten Bytes) auf verschiedenen Geräten gewährleistet werden

- Dokumentation
- Testbarkeit:

5) a)	Um im später erstellten Testfall (sowie für die geforderte Programmausgabe) einen Vergleichswert zu besitzen, wird eine String-Repräsentation der entschlüsselten Bytes gefordert. So kann der gegebene Plain mit der String-Repräsentation der entschlüsselten Bytes auf Gleichheit überprüft werden, um so zu erkennen ob die Ver- und Entschlüsselung korrekt durchgeführt wurden

- Anwendbarkeit:

1), 2), 4)	Auftretende Sicherheitslücken, etwa bezüglich der gewählten Algorithmen der Ver- und Entschlüsselungsinstanz sowie die gewählten Schlüssellängen, können laut der Webseite [Art] mittels statischer Analyse-Tools entdeckt werden
1)-5)	Das Vorhandensein aller einzelnen Bestandteile, sowie die richtige Repräsentation der Variablen können aber nur manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools

6.4 Richtlinien für digitale Signaturen

1. Generierung eines Schlüsselpaares
2. Erzeugung Signatur
   a) Initialisierung mit privatem Schlüssel
   b) Kodierung des Plaintexts beachten
   c) Nutzung eines sicheren Algorithmus zum Signieren
   d) Zusätzliche Erstellung der String-Repräsentation der Signatur
3. Validierung
   a) Validierung der erstellten Signatur mit öffentlichem Schlüssel

Abbildung 6.4: Visualisierung der Richtlinien für digitale Signaturen

Evaluation der Richtlinien anhand der Kriterien:

- Minimalität:

Nummer	Begründung
1)-3)	Die Richtlinien stellen die minimale Anzahl an Schritten für die Umsetzung des Anwendungsfalls für digitale Signaturen dar

- Sicherheit:

2)	Zur Auswahl der Algorithmen für die Erzeugung der Signatur sollte sich auf einer der in Kapitel 4.3 genannten Quellen bezogen sich in der Übersichtstabelle sicherer Algorithmen und Konzepte (siehe Tabelle 4.1) informiert werden

- Vollständigkeit:

1)-3)	Um den gesamten Anwendungsfall abzudecken muss erst ein Schlüsselpaar erzeugt, eine Signatur erstellt werden und anschließend diese validiert werden

- Kopierbarkeit:

2) b)	Durch die Beachtung der Kodierung des Plaintexts, soll ein einheitliches Resultat (hier der Signatur) auf verschiedenen Geräten gewährleistet werden

- Dokumentation
- Testbarkeit:

3)	Damit der Prozess des Signierens erfolgreich war, muss sich bei der Validierung ein Wahrheitswert von true ergeben. Der hier erhaltene Wert, kann in einem Testfall (für die Programmausgabe) mit dem erwarteten Wahrheitswert abgeglichen werden

- Anwendbarkeit:

2) c)	Auftretende Sicherheitslücken, etwa bezüglich der gewählten Algorithmen zur Generierung der Signatur, können laut der Webseite [Art] mittels statischer Analyse-Tools entdeckt werden
1)-3)	Das Vorhandensein aller einzelnen Bestandteile kann aber nur manuell als abhakbare Variante überprüft werden. Es besteht keine Möglichkeit zur Nutzung statischer Codeanalyse-Tools

7 Evaluation

In folgendem Kapitel sollen die im Laufe der Arbeit erstellten Richtlinien zur Evaluation zuerst auf konkrete Gesamtbeispiele angewendet werden. Danach soll überprüft werden, ob es mittels der Richtlinien möglich ist, korrekte (sowohl inhaltlich als auch strukturell) Gesamtbeispiele für die Webseite CryptoExamples zu erstellen.

7.1 Anwendung auf konkrete Beispiele

Im Folgenden werden die zur Erstellung notwendigen Richtlinien auf konkrete Beispiele angewendet. Aus Gründen der Übersichtlichkeit und damit der Code mit seinem Original besser verglichen werden kann, werden die Implementierungen der Beispiele als Abbildung dargestellt. Für jede Richtliniensorte (siehe Kapitel 4 - 6) wird eine eigene Abbildung verwendet. Die jeweiligen Punkte der Richtlinie werden im Beispiel markiert und mit der zugehörigen Nummer der originalen Richtlinie am linken Rand versehen. Es wurden dazu zwei Beispiele unterschiedlicher Sprachen (Java, Python) und Anwendungsfälle (Hashing, digitale Signatur) ausgewählt, um möglichst viele unterschiedliche sprachen- und anwendungsfall-abhängige Richtlinien abzudecken. Zwar wurde für die Sprache C# ebenfalls eine Richtlinie verfasst, jedoch konnte diese nicht weiter innerhalb dieses Kapitels evaluiert werden, da zum Erstellungszeitpunkt noch keine kompletten Beispiele dieser Sprache vorhanden waren.

Diese Anwendung der Richtlinien auf konkrete Beispiele dient zur Visualisierung wie die Richtlinien angewendet werden können und ebenfalls zur Überprüfung, ob die Richtlinien alle nötigen Angaben enthalten, um ein gesamtes Beispiel erstellen zu können oder ggf. festzustellen, ob der Richtlinie noch Aussagen zu bestimmten Bereichen fehlen. Außerdem kann dadurch überprüft werden, ob alle Richtlinienpunkte innerhalb des konkreten Beispiels auch eingehalten wurden. Da noch keine vollständige Prüfung aller bereits existierenden Beispiele auf der Webseite CryptoExamples erfolgt ist, fehlen teilweise noch Punkte, die innerhalb der Richtlinie jedoch gefordert werden. Die noch fehlenden oder nicht korrekt umgesetzten Punkte wurden unterhalb der jeweiligen Abbildung vermerkt.

Die folgenden Evaluationen werden auf bereits veröffentlichten Beispielen der Webseite CryptoExamples[1] vorgenommen (Abbildungen erstellt am 18.8.2018).

1 https://www.cryptoexamples.com/python_cryptography_string_signature_rsa.html
https://www.cryptoexamples.com/java_string_hash.html

7.1.1 Gesamtbeispiel: „Java String Hashing using JDK“

7.1.1.1 Allgemeine Richtlinien

Die nachfolgende Abbildung 7.1 stellt die allgemeinen Richtlinien der äußeren Dokumentation im Beispielcode „Java String Hashing using JDK“ dar.

Abbildung 7.1: Anwendung der Richtlinien zur äußeren Dokumentation (Kapitel 4.1)

Der Punkt 4) der Richtlinien zur äußeren Dokumentation (Kapitel 4.1) wurde hier nicht benötigt, da weder spezielle Voraussetzungen noch vorbereitende Maßnahmen für das Einbinden zusätzlicher Bibliotheken nötig waren.

Die Richtlinien werden vollständig erfüllt.

Die nachfolgende Abbildung 7.2 stellt die allgemeinen Richtlinien zur Implementierung im Beispielcode „Java String Hashing using JDK“ dar.

4)
```java
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.logging.Level;
import java.util.logging.Logger;
```

5) a
```java
/**
 * Example for hashing of a string in one method.
 * - SHA-512
 * - BASE64 encoding as representation for the byte-arrays
 * - UTF-8 encoding of String
 * - Exception handling
 */
```

6)
```java
public class ExampleHash {
  private static final Logger LOGGER = Logger.getLogger(ExampleHash.class.getName());
```

5) b
```java
  /**
   * Demonstrational method that hashes the plainText.
   * @param plainText
   * @return true if hashing was successful, false otherwise
   */
  public static boolean demonstrateHash(String plainText) {
```

7)
5) c
2)
```java
    try {
      // Get MessageDigest Instance
      MessageDigest messageDigest = MessageDigest.getInstance("SHA-512");

      // CREATE HASH
```

9) b
```java
      byte[] hashBytes = messageDigest.digest(plainText.getBytes(StandardCharsets.UTF_8));

      // CONVERT/ENCODE IN BASE64
```

5) c
9) a
```java
      String hashString = Base64.getEncoder().encodeToString(hashBytes);

      LOGGER.log(Level.INFO, hashString);
      return true;
```

8)
7)
```java
    } catch (NoSuchAlgorithmException e) {
      LOGGER.log(Level.SEVERE, e.getLocalizedMessage());
      return false;
    }
  }

  public static void main(String[] args) {
    demonstrateHash("Text that should be authenticated by comparing the hash of it!");
  }
}
```

Abbildung 7.2: Anwendung der Richtlinien zur Implementierung (Kapitel 4.2)

Der Punkt 1) der Richtlinien 4.2 kann nicht direkt im Beispiel sichtbar gemacht werden, da er sich auf die Version der Programmiersprache/Compiler bezieht.
Außerdem bietet Punkt 3) lediglich eine Anweisung wie die Bibliothek ausgesucht werden soll, weswegen er auch nicht in der Evaluation vermerkt werden konnte.

Die Richtlinien werden ansonsten vollständig erfüllt.

7.1.1.2 Sprachen-abhängige Richtlinien

Die nachfolgende Abbildung 7.3 stellt die sprachen-abhängigen Richtlinien im Beispielcode „Java String Hashing using JDK“ dar.

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Example for hashing of a string in one method.
 * - SHA-512
 * - BASE64 encoding as representation for the byte-arrays
 * - UTF-8 encoding of String
 * - Exception handling
 */
public class ExampleHash {
  private static final Logger LOGGER = Logger.getLogger(ExampleHash.class.getName());

  /**
   * Demonstrational method that hashes the plainText.
   * @param plainText
   * @return true if hashing was successful, false otherwise
   */
1) a,b 3)  public static boolean demonstrateHash(String plainText) {
    try {
      // Get MessageDigest Instance
      MessageDigest messageDigest = MessageDigest.getInstance("SHA-512");

      // CREATE HASH
      byte[] hashBytes = messageDigest.digest(plainText.getBytes(StandardCharsets.UTF_8));

      // CONVERT/ENCODE IN BASE64
      String hashString = Base64.getEncoder().encodeToString(hashBytes);

      LOGGER.log(Level.INFO, hashString);
1) c  return true;
    } catch (NoSuchAlgorithmException e) {
      LOGGER.log(Level.SEVERE, e.getLocalizedMessage());
      return false;
    }
  }

2), 3)
2) a  public static void main(String[] args) {
    demonstrateHash("Text that should be authenticated by comparing the hash of it!");
  }
}
```

Abbildung 7.3: Anwendung der sprachen-abhängigen Richtlinien (Kapitel 5)

Die Richtlinien werden vollständig erfüllt.

Die nachfolgende Abbildung 7.4 stellt die sprachen-abhängigen Richtlinien für Java im Beispielcode „Java String Hashing using JDK“ dar.

1)
```java
package com.cryptoexamples.java;


import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Example for hashing of a string in one method.
 * - SHA-512
 * - BASE64 encoding as representation for the byte-arrays
 * - UTF-8 encoding of String
 * - Exception handling
 */
```
2)
```java
public class ExampleHash {
```
3)
```java
  private static final Logger LOGGER = Logger.getLogger(ExampleHash.class.getName());

  /**
   * Demonstrational method that hashes the plainText.
   * @param plainText
   * @return true if hashing was successful, false otherwise
   */
```
4)
```java
  public static boolean demonstrateHash(String plainText) {
    try {
      // Get MessageDigest Instance
      MessageDigest messageDigest = MessageDigest.getInstance("SHA-512");

      // CREATE HASH
      byte[] hashBytes = messageDigest.digest(plainText.getBytes(StandardCharsets.UTF_8));

      // CONVERT/ENCODE IN BASE64
      String hashString = Base64.getEncoder().encodeToString(hashBytes);

      LOGGER.log(Level.INFO, hashString);
      return true;
    } catch (NoSuchAlgorithmException e) {
      LOGGER.log(Level.SEVERE, e.getLocalizedMessage());
      return false;
    }
  }

  public static void main(String[] args) {
    demonstrateHash("Text that should be authenticated by comparing the hash of it!");
  }
}
```

Abbildung 7.4: Anwendung der sprachen-abhängigen Richtlinien für Java (Kapitel 5.1)

Die Richtlinien werden bis auf einen Punkt vollständig erfüllt. Der Name der Klasse verletzt Punkt 2) b), da es sich beim zu nennenden Anwendungsfall-Name um „Hashing“ anstatt „Hash“ handeln würde.

7.1.1.3 Anwendungsfall-abhängige Richtlinien

Die nachfolgende Abbildung 7.5 stellt die anwendungsfall-abhängigen Richtlinien für Java im Beispielcode „Java String Hashing using JDK“ dar.

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Example for hashing of a string in one method.
 * - SHA-512
 * - BASE64 encoding as representation for the byte-arrays
 * - UTF-8 encoding of String
 * - Exception handling
 */
public class ExampleHash {
  private static final Logger LOGGER = Logger.getLogger(ExampleHash.class.getName());

  /**
   * Demonstrational method that hashes the plainText.
   * @param plainText
   * @return true if hashing was successful, false otherwise
   */
  public static boolean demonstrateHash(String plainText) {
    try {
      // Get MessageDigest Instance
1)    MessageDigest messageDigest = MessageDigest.getInstance("SHA-512");

      // CREATE HASH
2)    byte[] hashBytes = messageDigest.digest(plainText.getBytes(StandardCharsets.UTF_8));

      // CONVERT/ENCODE IN BASE64
3)    String hashString = Base64.getEncoder().encodeToString(hashBytes);

      LOGGER.log(Level.INFO, hashString);
      return true;
    } catch (NoSuchAlgorithmException e) {
      LOGGER.log(Level.SEVERE, e.getLocalizedMessage());
      return false;
    }
  }

  public static void main(String[] args) {
    demonstrateHash("Text that should be authenticated by comparing the hash of it!");
  }
}
```

Abbildung 7.5: Anwendung der der anwendungsfall-abhängigen Richtlinien für Hashing (Kapitel 6.1)

Die Richtlinien werden vollständig erfüllt.

Insgesamt werden bis auf einen Punkt alle Vorgaben der Richtlinien eingehalten. Alle Angaben, die zur Erstellung dieses Beispieles benötigt werden, sind des Weiteren durch die erstellten Richtlinien abgedeckt.

7.1.2 Gesamtbeispiel: „Python String Signing using Cryptography“

7.1.2.1 Allgemeine Richtlinien

Die nachfolgende Abbildung 7.6 stellt die allgemeinen Richtlinien der äußeren Dokumentation im Beispielcode „Python String Signing using Cryptography“ dar.

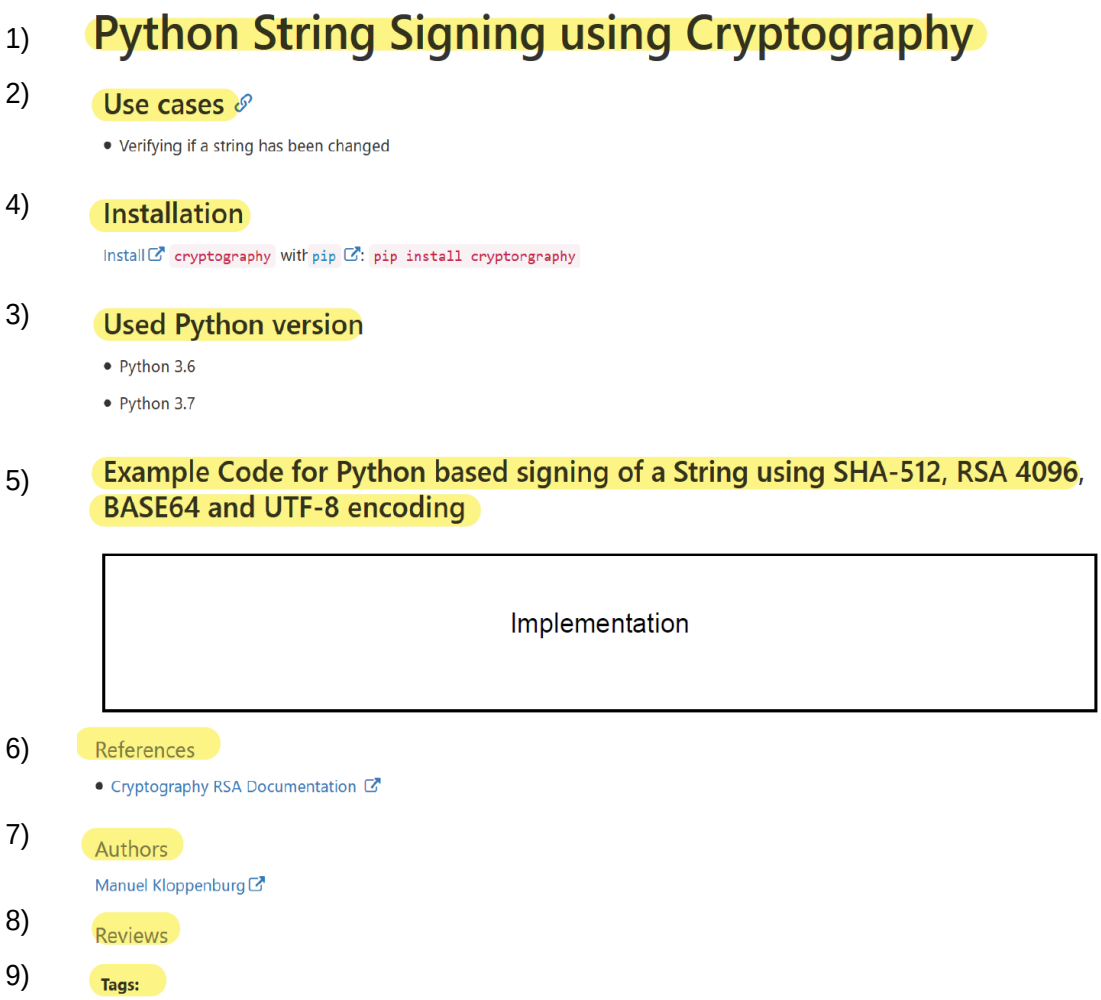

Abbildung 7.6: Anwendung der Richtlinien zur äußeren Dokumentation (Kapitel 4.1)

Da momentan noch nicht alle Beispiele der Webseite gegen die Richtlinien gereviewt wurden, fehlt diesem Beispiel noch die konkrete Tag-Angabe (Punkt 9)) der Richtlinien. Des Weiteren enthält der Beispiel-Name des Richtlinienpunktes 5) redundante Bestandteile („based signing of a String“) auf die auch verzichtet werden kann bzw. die nicht dem genannten Schema entsprechen.

Die nachfolgende Abbildung 7.7 stellt die allgemeinen Richtlinien zur Implementierung im Beispielcode „Python String Signing using Cryptography“ dar.

```
4)      import base64
        import logging

        from cryptography.exceptions import InvalidSignature
        from cryptography.exceptions import UnsupportedAlgorithm
        from cryptography.hazmat.backends import default_backend
        from cryptography.hazmat.primitives import hashes
        from cryptography.hazmat.primitives.asymmetric import padding
        from cryptography.hazmat.primitives.asymmetric import rsa

5) c    # set up logger
6)      logging.basicConfig(level=logging.INFO)
        logger = logging.getLogger(__name__)

        def demonstrate_signature_rsa(plain_text):
5) a        """
            All in one example for cryptographic signing of a string in one method.
            - Generation of public and private RSA 4096 bit keypair
            - SHA-512 with RSA signature of text using PSS and MGF1 padding
            - BASE64 encoding as representation for the byte-arrays
            - UTF-8 encoding of Strings
            - Exception handling
            """
7)          try:
5) c            # GENERATE NEW KEYPAIR
2)              private_key = rsa.generate_private_key(
2)                  public_exponent=65537,
                    key_size=4096,
                    backend=default_backend()
                )
                public_key = private_key.public_key()

5) c            # SIGN DATA/STRING
9) b            signature = private_key.sign(
                    plain_text.encode('utf-8'),
                    padding.PSS(
2)                      mgf=padding.MGF1(hashes.SHA512()),
                        salt_length=padding.PSS.MAX_LENGTH
2)                  ),
                    hashes.SHA512()

                )
9) a            logger.info("Signature: %s", base64.urlsafe_b64encode(signature))

5) c            # VERIFY JUST CREATED SIGNATURE USING PUBLIC KEY
7)              try:
9) b                public_key.verify(
                        signature,
                        plain_text.encode('utf-8'),
                        padding.PSS(
2)                          mgf=padding.MGF1(hashes.SHA512()),
2)                          salt_length=padding.PSS.MAX_LENGTH
                        ),
                        hashes.SHA512()
                    )
                    is_signature_correct = True
7)              except InvalidSignature:
                    is_signature_correct = False

6)              logger.info("Signature is correct: %s", is_signature_correct)
7)          except UnsupportedAlgorithm:
6)/7)           logger.exception("Signing failed")

        if __name__ == '__main__':
            # demonstrate method
            demonstrate_signature_rsa("Text that should be signed to prevent unknown tampering with its content.")
```

Abbildung 7.7: Anwendung der Richtlinien zur Implementierung (Kapitel 4.2)

Der Punkt 1) und 3) der Richtlinien zur Implementierung (Kapitel 4.2) kann wie bereits im Kapitel 7.1.1 erwähnt nicht direkt im Beispiel sichtbar gemacht werden. Ebenfalls ist die Art der momentanen Ausnahmebehandlung noch nicht mit den Richtlinienpunkten konform, da die Ausnahmen noch in mehreren Blöcken gefangen werden (Verletzung Punkt 7) a)). Da in Python keine Klassendefinition nötig ist kann auf die Kommentierung des Punktes 5) a) verzichtet werden.

7.1.2.2 Sprachen-abhängige Richtlinien

Die nachfolgende Abbildung 7.8 stellt die sprachen-abhängigen Richtlinien im Beispielcode „Python String Signing using Cryptography“ dar.

```
import base64
import logging

from cryptography.exceptions import InvalidSignature
from cryptography.exceptions import UnsupportedAlgorithm
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.asymmetric import rsa

# set up logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

1) a,b 3)

```
def demonstrate_signature_rsa(plain_text):
    """
    All in one example for cryptographic signing of a string in one method.
    - Generation of public and private RSA 4096 bit keypair
    - SHA-512 with RSA signature of text using PSS and MGF1 padding
    - BASE64 encoding as representation for the byte-arrays
    - UTF-8 encoding of Strings
    - Exception handling
    """
    try:
        # GENERATE NEW KEYPAIR
        private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=4096,
            backend=default_backend()
        )
        public_key = private_key.public_key()

        # SIGN DATA/STRING
        signature = private_key.sign(
            plain_text.encode('utf-8'),
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA512()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA512()

        )
        logger.info("Signature: %s", base64.urlsafe_b64encode(signature))

        # VERIFY JUST CREATED SIGNATURE USING PUBLIC KEY
        try:
            public_key.verify(
                signature,
                plain_text.encode('utf-8'),
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA512()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA512()
            )
            is_signature_correct = True
        except InvalidSignature:
            is_signature_correct = False

        logger.info("Signature is correct: %s", is_signature_correct)
    except UnsupportedAlgorithm:
        logger.exception("Signing failed")
```

2), 3)

```
if __name__ == '__main__':
    # demonstrate method
    demonstrate_signature_rsa("Text that should be signed to prevent unknown tampering with its content.")
```

2) a

Abbildung 7.8: Anwendung der sprachen-abhängigen Richtlinien (Kapitel 5)

Es fehlt Punkt 1) c) der sprachen-abhängigen Richtlinien (Kapitel 5). Ebenfalls finden sich im Methoden-Name Unterstriche, die laut Schema nicht nötig sind.

7.1.2.3 Anwendungsfall-abhängige Richtlinien

Die nachfolgende Abbildung 7.9 stellt die anwendungsfall-abhängigen Richtlinien für Python im Beispielcode „Python String Signing using Cryptography“ dar.

```
import base64
import logging

from cryptography.exceptions import InvalidSignature
from cryptography.exceptions import UnsupportedAlgorithm
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.asymmetric import rsa

# set up logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


def demonstrate_signature_rsa(plain_text):
    """
    All in one example for cryptographic signing of a string in one method.
    - Generation of public and private RSA 4096 bit keypair
    - SHA-512 with RSA signature of text using PSS and MGF1 padding
    - BASE64 encoding as representation for the byte-arrays
    - UTF-8 encoding of Strings
    - Exception handling
    """
    try:
        # GENERATE NEW KEYPAIR
1)      private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=4096,
            backend=default_backend()
        )
        public_key = private_key.public_key()

        # SIGN DATA/STRING
2)      signature = private_key.sign(
            plain_text.encode('utf-8'),
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA512()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA512()

        )
        logger.info("Signature: %s", base64.urlsafe_b64encode(signature))

        # VERIFY JUST CREATED SIGNATURE USING PUBLIC KEY
        try:
3)          public_key.verify(
                signature,
                plain_text.encode('utf-8'),
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA512()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA512()
            )
            is_signature_correct = True
        except InvalidSignature:
            is_signature_correct = False

        logger.info("Signature is correct: %s", is_signature_correct)
    except UnsupportedAlgorithm:
        logger.exception("Signing failed")


if __name__ == '__main__':
    # demonstrate method
    demonstrate_signature_rsa("Text that should be signed to prevent unknown tampering with its content.")
```

Abbildung 7.9: Anwendung der anwendungsfall-abhängigen Richtlinien für digitale Signaturen (Kapitel 6.4)

Diese Richtlinien werden vollständig erfüllt.

Insgesamt werden die Richtlinien in diesem Beispielcode nur unvollständig erfüllt. Jedoch zeigt sich auch hier, dass die Richtlinien alle Aussagen über relevante Bestandteile treffen, die zur Erstellung des Gesamtbeispiels notwendig sind.

7.2 Anwendung durch Testpersonen

Nachdem bereits im vorherigen Abschnitt 7.1 in der Theorie überprüft wurde, ob die Richtlinien alle notwendigen Angaben zur Erstellung eines konkreten Beispieles machen, soll dies nun des Weiteren auch praktisch überprüft werden. Dazu werden Testpersonen ausgewählt, die mit Hilfe der Richtlinien konkrete Beispiele erstellen sowie einen Fragebogen zur Richtlinien-Evaluation ausfüllen sollen. Dazu werden die Anwendungsfälle, die Sprache und die Bibliothek so gewählt, dass diese mit einem bereits erstellten Beispiel der Webseite übereinstimmen, sodass das Resultat der Testpersonen einfach mit dem vorhandenen Beispiel abgeglichen werden kann. Somit soll überprüft werden, ob die erstellten Richtlinien sich auch in der Praxis so benutzen lassen, dass als Ergebnis korrekte Beispiele für die Webseite erzeugt werden können. Neben der Erstellung des Beispiels selbst soll die Auswertung des Fragebogens ebenfalls dazu dienen die Aussagen der Richtlinien zu optimieren.

7.2.1 Durchführung

Für die Evaluation wurden fünf Testpersonen mit vorhandenen Programmiererfahrungen und keinem detaillierten Vorwissen über die Webseite CryptoExamples ausgewählt. In diesem Kontext diente diese Vorgabe der Programmiererfahrung lediglich dazu, dass die im Arbeitsauftrag geforderte Implementierung vorgenommen werden kann, was einer Testperson ohne Programmierkenntnissen nicht bzw. nur mit sehr hohem Aufwand möglich wäre. Die Formulierung des Arbeitsauftrages für die Testpersonen wurde in Zusammenarbeit mit Kai Mindermann verfasst und gestaltete sich dabei wie folgt:

Zur Umsetzung der geforderten Implementierung werden die Programmierumgebung Eclipse (4.8 Photon), ein aktuelles Java Development Kit (JDK) (Java SE 10.0.2) sowie die notwendigen Richtlinien zur Verfügung gestellt. Die Nutzung des Webs ist erlaubt. Ebenso die Installation weiterer Werkzeuge/Plugins etc.

1. **Aufgabe 1 (String Hashing):** Das Ergebnis dieser Aufgabe soll Java-Code sein, der das Hashen eines Strings demonstriert unter Verwendung der im JDK verfügbarer Funktionalität (ohne Nutzung anderer Crypto-Bibliotheken). Erstellen Sie den dazugehörigen Code unter Berücksichtigung der Ihnen vorliegenden Richtlinien.
2. **Aufgabe 2 (Symmetric String Encryption):** Das Ergebnis dieser Aufgabe soll Java-Code sein, der die Ver- und Entschlüsselung eines Strings demonstriert unter im JDK verfügbaren Funktionalität (ohne Nutzung anderer Crypto-Bibliotheken). Erstellen Sie den dazugehörigen Code unter Berücksichtigung der Ihnen vorliegenden Richtlinien.
3. **Aufgabe 3 (Ausfüllen des Fragebogens):** Füllen Sie den Fragebogen aus.

In der Aufgabenbeschreibung wurde darauf geachtet keine Hinweise auf die Webseite CryptoExamples und dessen GitHub-Projekt zu geben, da ansonsten der generelle Aufbau der Beispiele bereits eingesehen werden könnte. Die Nutzung der Webseite während des Tests musste demnach ebenfalls vermieden werden. Dies wurde zum einen durch eine

Begleitung der Evaluation als auch durch die Sperrung der entsprechenden Webseiten im zur Verfügung gestellten Browser sichergestellt.

Der Arbeitsauftrag wurde dabei als ausgedruckte Version und die Richtlinien als html-Datei geöffnet im Browser zur Verfügung gestellt. Ein Zeitlimit für die Bewältigung der Aufgaben wurde nicht gestellt, da die Programmiererfahrung sowie verschiedene Kenntnisse bzgl. der Kryptografie zur unterschiedlichen Schnelligkeit der Testpersonen beitragen.

Die Richtlinie der äußeren Dokumentation (Kapitel 4.1) wurden dabei außer Acht gelassen, da hier generell überprüft werden soll, ob die Testpersonen, anhand der gegebenen Richtlinien, in der Lage sind die Implementierung des Anwendungsfalls korrekt durchzuführen. Nach Beendigung der Implementierungen folgte ein Fragebogen, der sich in zwei Bereiche („Vorkenntnisse, Erlebnis im Experiment und Demographie“ und „Security APIs“) unterteilt. Im ersten Block wurden persönliche Daten wie Programmier- und Kryptografie-Kenntnisse und die konkreten Erlebnisse im Experiment festgehalten. Auch eine Bewertung der Richtlinien wurde durch konkrete Fragen aufgezeichnet. Dadurch soll die subjektive Meinung der Testpersonen berücksichtigt werden, um somit die Richtlinien nach der Analyse besser optimieren zu können. Ebenfalls besteht dadurch die Möglichkeit zu vergleichen, ob gefundene Probleme bei der Analyse der erstellten Implementierung mit der Bewertung der Richtlinien der Testpersonen konform sind. Ob tatsächlich auch alle Richtlinien-Punkte korrekt umgesetzt und damit verstanden wurden, wird jedoch am Ende beim Abgleich mit dem Original-Beispiel der Webseite überprüft. Der zweite Block basiert auf der Quelle [WAS17] und enthält Fragen zur verwendeten Security API (hier die bereitgestellten Methoden des JDKs im Paket javax.crypto.* bzw. java.security.*) wie Handhabung und Erfahrung der Testpersonen im Zusammenhang mit dieser während des Tests. Wobei dieser Fragenblock nur zusätzlich festgehalten und gesondert innerhalb eines anderen Kontextes genauer analysiert wird, da dies nicht direkt mit der Evaluation der Richtlinien zusammenhängt.

Die von den Testpersonen erstellten Implementierungen wurden anschließend gegen die Richtlinienpunkte abgeglichen, um so festzustellen, welche eingehalten oder nicht bzw. falsch umgesetzt wurden. Des Weiteren können die Implementierungen gleicher Anwendungsfälle aller Testpersonen nochmals gegeneinander abgeglichen werden, um damit zu analysieren, ob alle identische Probleme hatten. Aus diesem Grund ist es also sinnvoll, alle Testpersonen die gleichen Anwendungsfälle in derselben Sprache implementieren zu lassen um häufige Fehler besser erkennen und beheben zu können. Ebenso wurden die Antworten des Fragebogens ausgewertet und auch im Zusammenhang mit den erstellten Implementierungen betrachtet.

7.2.2 Auswertung

Bei den fünf Testpersonen handelte es sich um zwei weibliche und drei männliche Teilnehmer mit einem Durchschnittsalter von 22 Jahren. Dabei sind alle davon in einem informationstechnischen Studiengang (Informatik, Softwaretechnik, Medieninformatik) eingeschrieben. Lediglich eine Testperson hat nebenbei einen Job in der Softwareentwicklung. Die selbst eingeschätzten Programmierkenntnisse lagen (Skala von 0 (niedrig) - 10 (hoch))

bei durchschnittlich 5 Punkten. Die Kenntnisse im Bereich der Kryptografie und Informationssicherheit bei 4 Punkten. Keine der Testpersonen hat schon einmal mit einer anderen Kryptografie-Bibliothek gearbeitet. Die einzelnen Fragen des Fragebogens sowie die Antworten der Testpersonen können im Anhang (siehe Anhang A.1) genauer betrachtet werden.

Bei der Beobachtung des Vorgehens der Testpersonen während der Erstellung der Implementierungen, war besonders auffällig, dass drei der fünf Testpersonen zuerst damit begonnen haben zu implementieren ohne die allgemeinen oder sprachen-abhängigen Richtlinien zu beachten. Erst als die Implementierung alle ihrer Meinung nach notwendigen Funktionalitäten beinhaltete wurde der entstandene Code anhand der genannten Richtlinien angepasst. Zwar wurden die allgemeinen und sprachen-abhängigen Richtlinien nicht sofort von den Testpersonen umgesetzt, die anwendungsfall-abhängigen Richtlinien wurden jedoch gleich zu Beginn mitberücksichtigt. Vor der eigentlichen Implementierung eines Anwendungsfalls wurde zuerst der umzusetzende Anwendungsfall in den entsprechenden Richtlinien gesucht und anschließend die Tabelle für zu verwendende Algorithmen und Konzepte (siehe Tabelle 4.1) betrachtet. Von allen Testpersonen wurde die Tabelle als positiv bewertet. Lediglich zwei der fünf Teilnehmer haben bereits nach der Nutzung der anwendungsfall-abhängigen Richtlinien, auch die allgemeine und sprachen-abhängige Richtlinien während der Implementierung genutzt, um nachzuschauen wie bestimmte Bereiche gehandhabt werden sollen.

In der bereitgestellten Version der Richtlinien kamen zuerst die allgemeinen, dann die anwendungsfall- und zuletzt die sprachen-abhängigen Richtlinien. Anhand der vorherigen Beobachtung zum Umgang der Testpersonen mit den Richtlinien, würde die optimale Reihenfolge jedoch zuerst die anwendungsfall-abhängigen, dann die sprachen-abhängigen und zuletzt die allgemeinen Richtlinien umfassen. Zuerst wurde stets die zugehörige anwendungsfall-abhängige Richtlinie sowie die Übersichtstabelle angeschaut und anschließend im Web recherchiert. Danach wurde das Schema für die Klassen- und Methoden-Namen in den sprachen-abhängigen Richtlinien relevant und am Ende Dinge wie die genaue Umsetzung der Fehlerbehandlung oder Kommentierung der allgemeinen Implementierungs-Richtlinie. Somit kann durch die Anpassung der Reihenfolge ebenfalls der Arbeitsablauf der künftigen Beispielersteller unterstützt werden.

Insgesamt wurden bei der Auswertung des Fragebogens von den Testpersonen kein Bereich angemerkt, der ihnen zur Erstellung des Beispiels von den Richtlinien gefehlt hätte. Auch auf Nachfrage, ob spezielle Richtlinien-Punkte zu ungenau formuliert waren, wurde lediglich von einer Testperson auf eine konkrete Aussage verwiesen, auf welche innerhalb des Kapitels noch genauer eingegangen wird. Allgemein schienen die Richtlinien-Punkte also nicht zu abstrakt oder unklar formuliert worden zu sein. Jedoch wurde bei der Analyse der erstellten Implementierungen deutlich, dass einige Richtlinien-Punkte (zur tatsächlich gewünschten Umsetzung) das Einfügen von zusätzlichen Informationen benötigen. Trotzdem liegt die Einschätzung der Testpersonen bzgl. des Sicherheitsaspektes ihrer eigenen Implementierung durchschnittlich bei nur 3 Punkten. Diese Unsicherheit kann evtl. auf die relativ geringe Erfahrung der Testpersonen (durchschnittlich 4 Punkte) zurückgeführt werden.

Es werden nun die verschiedenen Richtlinien (allgemein (Kapitel 4.2), sprachen-abhängig (Kapitel 5), anwendungsfall-abhängig (Kapitel 6)) anhand der Ergebnisse der Evaluation (Fragebogen, Implementierungen) bewertet und Verbesserungsmöglichkeiten angegeben. Der Anhang A.2 zeigt eine Übersicht, die visualisiert welche konkreten Richtlinien-Punkte von den einzelnen Testpersonen jeweils umgesetzt oder nicht beachtet wurden.

Allgemeine Richtlinien - Implementierung

Betrachtet man besonders die entstanden Implementierungen der Aufgabe 1 (Hashing) fällt auf, dass bezüglich der ursprünglichen Richtlinien-Aussage: „Byte-Arrays werden für die Ausgabe kodiert“ (siehe Richtlinie Kapitel 4.2 Punkt 9) bzw. „Erstellung einer String-Repräsentation des Hashs“ (siehe Richtlinie Kapitel 6.1 Punkt 3) weiter präzisiert werden sollte. Bei den Implementierungen der fünf Testpersonen gab es drei unterschiedliche Weisen wie die Umwandlung des Byte-Arrays zu einem String erfolgte. Je nach Vorgehen werden entsprechend auch für denselben Plain unterschiedliche Hashwerte ausgegeben, was nicht der Fall sein sollte. Somit sollte in den angesprochenen Punkten zusätzlich z.B. auf die Nutzung von Base64 als Kodierungsverfahren hingewiesen werden, um somit einheitlich für denselben Plain identische Hashwerte zu gewährleisten. Auch in Aufgabe 2 wurde von den Testpersonen unterschiedliches, teilweise umständliches Vorgehen zum Erstellen einer String-Repräsentation benutzt, was die Notwendigkeit der Anpassung ebenfalls unterstützt.

Ein weiteres Problem stellte der Punkt: „Strings werden mit UTF-8 kodiert“ (siehe Richtlinie Kapitel 4.2 Punkt 9) und damit auch die „Erstellung der String-Repräsentation der entschlüsselten Bytes“ (siehe Richtlinie Kapitel 6.2.1 Punkt 4, 6) dar. Zwar wurde der Punkt von allen Testpersonen bei der Umwandlung eines Byte-Arrays zu einem String (.getBytes(„utf-8“)) beachtet, jedoch wurde dies nicht durchgängig in der gesamten Implementierung bedacht. So wurde die Kodierung etwa bei der Erstellung eines neuen Strings (new String(...)), nicht mehr mit angegeben. Dies liegt also nicht an Missachtung der Richtlinien-Aussage, sondern an eventuell fehlendem Wissen, wo die Kodierung UTF-8 mit angegeben bzw. beachtet werden muss. Dies kann durch einen zusätzlichen Hinweis (z.B. in Klammern) zur Beachtung der Kodierung UTF-8 des Strings direkt hinter den Punkten, die eine String-Repräsentation fordern, geschehen. Dadurch wird dies nochmals hervorgehoben bzw. wird dem Vergessen vorgebeugt.

Im Kontext der Kommentierung wird eine „Funktionale Beschreibung und Auflistung verwendeter Algorithmen und Konzepte des folgenden Codes“ innerhalb der Klassendokumentation (siehe Richtlinie Kapitel 4.2 Punkt 5) gefordert. Bisher bedeutete dies bei fertigen Beispielen auf der Webseite CryptoExamples, dass dort auch die genutzten Kodierungsverfahren (wie Base64 und UTF-8) angegeben wurden. Jedoch wurden diese von keiner der Testpersonen auch automatisch als Algorithmus oder Konzept aufgefasst, wie der Richtlinien-Punkt in seiner ursprünglichen Form gefordert hatte. Sollen sie also angegeben werden muss dies explizit dem Richtlinien-Punkt hinzugefügt werden.

Ein Punkt, der explizit als unklar formuliert von einer Testperson im Zuge des Fragebogens genannt wurde, war: „Ausnahmen werden am Ende abgefangen (um den Rest des Codes nicht zu überladen)“ (siehe Richtlinie Kapitel 4.2 Punkt 7 a). Gewünscht wurde

hier die Implementierung eines Blocks (hier in Java: ein try-catch-Block um den gesamten Code der Methode), der alle Ausnahmen, die vom gesamten Code geworfen werden können, abfängt. Von der Person, die diesen Punkt als unklar angab, wurde die Implementierung zwar trotzdem korrekt umgesetzt und damit interpretiert, jedoch wurde der Punkt von einer anderen Testperson tatsächlich nicht wie gedacht umgesetzt. Der Punkt wurde als Abfangen beim Methoden-Aufruf in der Main-Methode interpretiert und die Ausnahmen von der Methode geworfen. Dies zeigt, dass diese Aussage genauer formuliert werden muss. Um dem entgegen zu wirken könnte auch innerhalb der sprachen-abhängigen Richtlinien für Java ein konkreter Punkt zur Ausnahmen-Behandlung verfasst werden, der das Abfangen in einem try-catch-Block nochmals genau erklärt, womit der Punkt nicht mehr so abstrakt und damit unklar wirkt.

Ebenfalls auffällig waren die Ergebnisse zum Richtlinien-Punkt 8 Kapitel 4.2 bzgl. der Programmausgabe. Dieser wurde zwar von allen bedacht, jedoch waren die Ausgaben, die dafür durch den Logger ausgegeben wurden, unterschiedlich. So wurden etwa bei der symmetrischen Verschlüsselung von drei Testpersonen lediglich der zu verschlüsselnde Text und die wieder entschlüsselte Cipher ausgegeben, was zeigt, dass die Verschlüsselung erfolgreich war. Die anderen beiden Testpersonen haben diese beiden Strings nochmal explizit verglichen und somit einen Boolean-Wert ausgegeben, was ebenfalls korrekt die Funktionalität beweist. Falls explizit eine Ausgabe eines bestimmten Wertes gewünscht ist sollte dies im jeweiligen Anwendungsfall nochmals genauer konkretisiert werden. In diesem Kontext variierten auch die Rückgabe-Werte und Typen der Methode (siehe Richtlinie Kapitel 5.1 Punkt 4 b).

Sprachen-abhängige Richtlinien

Insgesamt fällt sowohl bei den Implementierungen der Aufgabe 1 als auch bei denen der Aufgabe 2 auf, dass die bisherigen Aussagen bzgl. der Namenskonventionen wie: „Name: demonstrate$Anwendungsfall-Name“ (siehe Richtlinie Kapitel 5 Punkt 1) und „Name: Example$Anwendungsfall-Name“ (siehe Richtlinie Kapitel 5.1 Punkt 2) von zwei Testpersonen nicht wie gedacht umgesetzt wurden. Es kam dabei auch direkt während der Durchführung des Tests die Frage auf was genau für den Anwendungsfall-Namen eingesetzt werden sollte. So wurden etwa der konkrete genutzte Algorithmus oder lediglich ein Teil des Namens von den Testpersonen als Anwendungsfall-Name interpretiert (Angabe der Testpersonen z.B. „ExampleSHA256“, „ExampleEncryption“, gewünscht: „ExampleHashing“, „ExampleSymmetricEncryption“). Der Anwendungsfall-Name sollte dabei aus den anwendungsfall-abhängigen Richtlinien entnommen werden. Diese Richtlinien besitzen auch bei der Darstellung auf GitHub als Überschrift ihren jeweiligen Anwendungsfall-Namen, der in der Namenskonvention genutzt werden sollte. Zur Behebung dieses Problems könnte also auf die Überschriften der Anwendungsfall-abhängigen Richtlinien verwiesen oder zumindest ein konkretes Beispiel zusätzlich zu der Angabe des eigentlichen Namens-Schemas gegeben werden.

Ansonsten wurden die Punkte der sprachen-abhängigen Richtlinien (siehe Kapitel 5) von den Testpersonen wie gewünscht eingehalten.

Anwendungsfall-abhängige Richtlinien

Die Richtlinien des jeweiligen Anwendungsfalls wurden von den Testpersonen jeweils komplett eingehalten. Bei der Erfüllung der Aufgabe 1 kam es jedoch vor, dass zusätzliche Funktionalitäten, die über die Richtlinien-Punkte hinausgehen, eingebaut wurden. Beim Hashing fordert die Richtlinien lediglich die einfachste Umsetzung des Anwendungsfalles, in dem einfach eine gewählte sichere Hashfunktion auf den Plain angewendet wird. Zusätzlich wurde jedoch an den Plain ein gewählter String für den Salt mit gehasht. Dies ist für den Sicherheitsaspekt des Hashs nicht von Nachteil, war jedoch von der Richtlinie nicht gefordert. Soll dies also unterbunden werden kann ebenfalls explizit angegeben werden, dass die zusätzliche Beachtung eines Salt-Wertes nicht notwendig ist.

Ein weiteres Problem stellte bei Aufgabe 2 der Punkt: „Auswahl des zu verwendenden Algorithmus, der Authenticated Encryption (AE) durchführt“ dar. Drei der Testpersonen nutzten für die Verschlüsselung lediglich AES mit der korrekten Schlüssellänge, jedoch stellt dies kein AE-Verfahren dar und unterstützt somit nicht die Integrität und ist somit ebenfalls laut Webseite [Art] nur unzureichend sicher. Dies könnte evtl. dadurch entstanden sein, dass zwar die entsprechenden Algorithmen in der mit vorhanden Übersichtstabelle (Tabelle 4.1) nachgeschaut wurden, diese aber eine Zeile für die Modis und eine für die Blockchiper enthielt. Dies sollte so genutzt und gelesen werden, dass eine sichere Blockchiper in Kombination mit einem sicheren Modus benutzt wird. Jedoch könnte dies dazu geführt haben, dass die Nutzung eines sicheren Modus als ausreichend angesehen wurde. Somit sollte in der Übersichtstabelle zusätzlich vermerkt werden, dass die Modis in Kombination mit einer Blockcipher (welche AE umsetzt) zu betrachten sind. Dieses Vorgehen führte ebenfalls dazu, dass in der Implementierung keine Nonce mehr benötigt wurde, welche ansonsten mittels eines Zufallszahlengenerators hätte erstellt werden müssen.

Außerdem wurden von zwei der Testpersonen die vorhandenen Abbildungen (siehe Abbildung 6.1, 6.2), die den Prozess des jeweiligen Anwendungsfalls abbilden (unterhalb der anwendungsfall-abhängigen Richtlinien), als positiv und hilfreich bewertet. Auffällig war hier, dass die Testpersonen, die zur Erstellung neben den eigentlichen Aussagen der anwendungsfall-abhängigen Richtlinien auch die zugehörigen Abbildungen betrachteten, auch diejenigen waren, die den Anwendungsfall der symmetrischen Verschlüsselung als einzige korrekt mit AE-Verfahren und damit sicher umgesetzt haben, während die anderen Testpersonen den Abbildungen wenig Beachtung entgegenbrachten. Die Abbildungen scheinen also als Leitlinie zur Umsetzung gut geeignet zu sein und könnten auch zu einer korrekteren, sichereren Umsetzung beitragen.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Im Laufe der Bachelorarbeit wurden Richtlinien verfasst, die den gesamten Arbeitsprozess zur Erstellung eines Codebeispiels für CryptoExamples umfassen. Dabei machen diese Aussagen über allgemeine, sprachen-/anwendungsfall-unabhängige sowie sprachen- und anwendungsfall-abhängige Punkte. Die Richtlinien-Inhalte machen dabei mindestens Aussagen über die Kriterien Minimalität, Sicherheit, Vollständigkeit, Kopierbarkeit, Dokumentation, automatische Testbarkeit und Anwendbarkeit. Im Zuge dessen wurde eine jeweilige Evaluation der Richtlinienpunkte gegen die genannten Kriterien vorgenommen.

Um sowohl die Richtlinien als auch die bereits existierenden Beispiele auch im Weiteren Betrieb der Webseite aktuell zu halten wurde ebenfalls der notwendige Überprüfungsprozess beschrieben und in diesem Zusammenhang deren Lebenszyklus erstellt und untersucht. Außerdem wurde eine Übersichtstabelle erstellt, die die Ermittlung sicherer Algorithmen und Konzepte erleichtern und unterstützen soll, um den Sicherheitsaspekt der Codebeispiele zu gewährleisten.

Als die Richtlinien fertig gestellt waren, wurden diese evaluiert indem sie auf konkrete Beispiele der Webseite angewendet wurden. Ebenfalls wurden Codebeispiele von Testpersonen anhand der angefertigten Richtlinien erstellt und ein Fragebogen ausgefüllt. Anhand dieser wurden die Richtlinien kritisch betrachtet und Verbesserungsvorschläge erarbeitet.

In Zeiten der Sozialen Medien, der Verlagerung der Finanzverwaltung ins Internet und der zunehmenden Digitalisierung bleibt die Sicherheit privater Daten zukünftig ein sehr wichtiger Aspekt.

8.2 Ausblick

Mit der Definition und Evaluation der Richtlinien wurde eine Basis geschaffen, um die Pflege der Codebeispiele auf CryptoExamples sowie eine Erweiterung um weitere Beispiele, durch eine breite Basis an aktiven Unterstützern zu gewährleisten. Auch in Zukunft wird eine Weiterentwicklung und Anpassung der Richtlinien notwendig sein. Die Ansprüche können sich im Laufe der Zeit ändern und Richtlinien für neue Sprachen und/oder Anwendungsfälle können notwendig werden, was eine regelmäßige Prüfung und Anpassung notwendig macht. Dazu bietet der Lebenszyklus der Richtlinien und der Codebeispiele notwendige Anhaltspunkte. Trotz der zentralen Rolle der Richtlinien wird die Bekanntheit der Webseite

von ihrer Auffindbarkeit im Internet, von der Anzahl der Codeersteller und deren Engagement, sowie ihrer Langlebigkeit und dem Sicherheitsniveau der Codebeispiele abhängen.

Ebenfalls muss sich weiter um genauere Aussagen bezüglich der Schlüsselverwaltung gekümmert werden, da im Umfang dieser Arbeit bisher keine Richtlinien dieses Anwendungsfalls verfasst wurden, da der sensitive Umgang mit Schlüsseln vor allem in der Kryptografie von großer Bedeutung ist. Werden die Schlüssel etwa nicht richtig gespeichert, könnte dies zu Zugriffen nicht berechtigter Parteien beitragen. Daher ist für diesen Bereich zusätzliche, umfangreiche und sorgfältige Recherche notwendig, bevor genauere Aussagen, auch innerhalb von zukünftigen Richtlinien, verfasst werden sollen. Ebenfalls wird in Zukunft auch der Prozess der Dateiverschlüsselung näher betrachtet werden.

Um die Webseite zu erweitern wird momentan an Codebeispielen in Sprachen wie C#, JavaScript oder Rust (auch im Umfang weiterer Bachelorarbeiten) gearbeitet, welche bei Fertigstellung der Webseite hinzugefügt werden.

A Anhang

A.1 Fragebogen-Auswertung

Fragebogenbereich “Vorkenntnisse, Erlebnis im Experiment und Demographie“

Einzelne Bewertungen erfolgten im Zahlenbereich 0 (niedrig) bis 10 (hoch).

Waren bestimmte Aussagen der Richtlinien unverständlich bzw. zu ungenau formuliert? Fügen Sie die entsprechende Aussage ein (Copy/Paste) und schlagen Sie eine Verbesserung vor.
Nein Nein [ ] Exceptions are caught at the end (to not clutter the rest of the code) Hier könnte man direkt schreiben, dass ein try catch Block verlangt wird. / Nein

Haben Sie Aussagen in den Richtlinien vermisst die Ihnen geholfen hätten? Wenn ja, welche?
Nein Nein / Evtl. Links zu Quellen, bei denen schon (sicherer) code implementiert zu finden ist? Nein

Haben Sie die bereitgestellte Entwicklungsumgebung (Eclipse) mit zusätzlichen Werkzeugen erweitert (Plugins installiert) oder andere Software als Eclipse, Java-Compiler und den Browser zur Lösung der Aufgaben verwendet? Wenn ja, welche?
Nein Nein / / Nein

Geschlecht
weiblich männlich weiblich männlich männlich

Wie alt sind Sie?
21 23 20 22 22

Was ist ihr derzeit angestrebter Bildungsabschluss bzw. falls Sie keinen weiteren Bildungsabschluss anstreben was ist ihr bisheriger höchster Bildungsabschluss? (angestrebt/erreicht, Fachrichtung/Studiengang und Akademischen Grad nennen)
B.Sc Informatik B.Sc. Informatik angestrebt: Master of Science, Medieninformatik B.S. angestrebt: Bachelor of Science B.Sc. Softwaretechnik

Wie schätzen Sie ihre Programmiererfahrung ein?
4
5 (Durchschnittliche Erfahrung)
4
6
7

Verdienen Sie neben dem Studium als Softwareentwickler oder Programmierer Geld?
Nein
Nein
Nein
Ja
Nein

Wie schätzen Sie Ihre Kenntnisse in Bezug zu Security und Kryptographie ein?
6
4
4
5 (Durchschnittliche Kenntnisse)
3

Haben Sie schonmal eine andere Kryptographie-Bibliothek verwendet, wenn ja welche?
Nein
Nein
bcrypt
/
Nein

Haben Sie sich speziell für das heutige Experiment in irgendeiner Form vorbereitet, wenn ja wie ausführlich und in welcher Form?
Nein
Nein
Nein
Nein
Nein

Das Niveau der Aufgabe war für mich
genau richtig
genau richtig
etwas zu hoch (war ziemlich anstrengend)
genau richtig
genau richtig

Wie bewerten Sie die Sicherheit Ihres programmierten Beispiels?
4
3
4
3
3

Kommentare zum Experiment, zur Aufgabe, zu Kryptographie in Java oder sonstigem:
/
/
Bis jetzt kannte ich nur die theoretischen Grundlagen der Kryptographie. Durch diese Studie durfte ich zum ersten Mal eine symmetrische Verschlüsselung implementieren. Die Guidelines waren sehr hilfreich.
Die zweite Aufgabe war einfacher als die erste
/

A.2 Umsetzung der Richtlinien-Punkte durch die Testpersonen

Implementation / Example Code	P1	P2	P3	P4	P5
Compliant with current coding guideline					
Code can be executed with latest stable version of the programming language/ Code can be executed with the latest stable version of the common tool chain of the programming language					
Only uses algorithms and concepts that are secure					
Import/Using statements are explicit					
Program output is made through a logging-facility (logger) and not via unfiltered system					
Exceptions are caught except runtime exceptions					
Exceptions are caught at the end					
Exceptions are logged via the logging facility					
No stack-trace is printed					
The code demonstrates how its security functionality can be evaluated					
Byte arrays are encoded for output					
Strings are encoded using UTF-8					
Class Documentation describes functionality/ lists used algorithms and concepts					
Method Documentation describes functionality, parameters and return value					
Inline comments describe essential parts of the code					

Language specific guidelines	P1	P2	P3	P4	P5
Class named class like: Example$SCENARIO					
Example code method accepts appropriate parameter (usually String)					
The example only contains one defined method for the example demonstration AND a main method.					
Main method calls the demonstrator method with parameter					
Return a result depending on the application output					
Defined example code method like: demonstrate$SCENARIO					
Class is “public”					
Method is “public”					

Scenario specific guidelines	P1	P2	P3	P4	P5
Hashing:					
Created or accepted a string to be hashed					
Regarded the encoding of the string					
Created the hash					
Encoded the hash to be represented as string					
Symmetric encryption:					
Created or accepted a plaintext string to be encrypted					
Used a cryptographically suitable random number generator					
Used an authenticated encryption algorithm					
Regarded the encoding of the string					
Encrypted the plaintext					
Encoded the ciphertext to be represented as string					
Decrypted the ciphertext					
Encoded the decrypted bytes to be represented as string					

beachtet und umgesetzt

nicht beachtet oder nicht umgesetzt

Literaturverzeichnis

[And93] R. Anderson. „Why cryptosystems fail“. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. ACM. 1993, S. 215–227 (zitiert auf S. 12).

[Art] P. Arteau. *Bug Patterns*. Hrsg. von Find Security Bugs. URL: https://find-sec-bugs.github.io/bugs.htm (zitiert auf S. 11, 15, 29, 47–49, 51, 53, 69).

[Bar16] E. Barker. *Recommendation for Key Management - Part 1: General*. Hrsg. von NIST Special Publication 800-57 Part 1 Revision 4. 2016. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf (zitiert auf S. 11, 32, 33).

[BD15] E. Barker, Q. Dang. *Recommendation for Key Management - Part 3: Application-Specific Key Management Guidance*. Hrsg. von NIST Special Publication 800-57 Part 3. 2015. URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-57pt3r1.pdf (zitiert auf S. 33).

[Buc01] J. Buchmann. *Einführung in die Kryptographie*. Bd. 5. Springer, 2001 (zitiert auf S. 3, 4).

[Bun18] Bundesamt für Sicherheit in der Informationstechnik. *Cryptographic Mechanisms: Recommendations and Key Lengths*. 2018. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=7 (zitiert auf S. 11, 14, 15, 32, 33).

[CER] CERT. *SEI CERT Coding Standards*. Hrsg. von Carnegie Mellon University - Software Engineering Institute. URL: https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards (zitiert auf S. 11).

[Dwo11] M. Dworkin. *Recommendation for Block Cipher Modes of Operation Methods and Techniques*. Hrsg. von NIST Special Publication 800-38A 2001 Edition. 2011. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf (zitiert auf S. 33).

[Fas] FastWP-Webseite. *Was ist Base64 und was bringt es?* URL: https://fastwp.de/4810/ (zitiert auf S. 30).

[Gau12] P. Gauravaram. „Security Analysis of salt || password Hashes“. In: *Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on*. IEEE. 2012, S. 25–30 (zitiert auf S. 15).

[GFN+17] P. Grassi, J. Fenton, E. Newton, R. Perlner, A. Regenscheid, W. Burr, J. Richer. *Digital Identity Guidelines-Authentication and Lifecycle Management*. Hrsg. von NIST Special Publication 800-63B. 2017. URL: https://pages.nist.gov/800-63-3/sp800-63b.html#sec5 (zitiert auf S. 33).

[Gir18] D. Giry. *BlueKrypt*. 2018. URL: https://www.keylength.com/en/compare/ (zitiert auf S. 33).

[Gut02] P. Gutmann. „Lessons Learned in Implementing and Deploying Crypto Software“. In: *Usenix Security Symposium*. 2002, S. 315–325 (zitiert auf S. 11).

[Ins18] Institut der Informationssicherheit - Universität Stuttgart. *Vorlesung: Grundlagen der Informationssicherheit*. WS 17/18 (zitiert auf S. 14–16).

[Jav17] Javarevisited-Webseite. *10 Best Practices to Follow While Writing Code Comments*. 2017. URL: https://javarevisited.blogspot.com/2011/08/code-comments-java-best-practices.html (zitiert auf S. 28, 30).

[NSMB12] S. M. Nasehi, J. Sillito, F. Maurer, C. Burns. „What makes a good code example?: A study of programming Q&A in StackOverflow“. In: *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE. 2012, S. 25–34 (zitiert auf S. 12).

[NSS] Y. Nir, R. Salz, N. Sullivan. *Transport Layer Security (TLS) Parameters*. URL: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4 (zitiert auf S. 15, 33).

[Ora] Oracle-Webseite. *How to Write Doc Comments for the Javadoc Tool*. URL: http://www.oracle.com/technetwork/articles/java/index-137868.html (zitiert auf S. 30).

[Pyt] Python Software Foundation-Webseite. *Python Documentation by Version*. URL: https://www.python.org/doc/versions/ (zitiert auf S. 36).

[Rus18] N. Rusam. „Erstellung von CryptoExamples in C#“. 2018 (zitiert auf S. 40).

[Sea] R. Seacord. *Top 10 Secure Coding Practices*. Hrsg. von Carnegie Mellon University - Software Engineering Institute. URL: https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices (zitiert auf S. 11, 28).

[TBBC10] M. Turan, E. Barker, W. Burr, L. Chen. *Recommendation for Password-Based Key Derivation*. Hrsg. von NIST Special Publication 800-132. 2010. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf (zitiert auf S. 15, 33, 48, 49).

[Thea] The Open Web Application Security Project (OWASP). *Cryptographic Storage Cheat Sheet*. URL: https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet (zitiert auf S. 47, 49).

[Theb] The Open Web Application Security Project (OWASP). *Logging Cheat Sheet*. URL: https://www.owasp.org/index.php/Logging_Cheat_Sheet (zitiert auf S. 28, 29).

[Thec] The Open Web Application Security Project (OWASP). *OWASP Secure Coding Practices Checklist*. URL: https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_Checklist (zitiert auf S. 11, 29).

[Thed] The Open Web Application Security Project (OWASP). *Secure Coding Cheat Sheet*. URL: https://www.owasp.org/index.php/Secure_Coding_Cheat_Sheet (zitiert auf S. 11).

[Thee] The Open Web Application Security Project (OWASP). *Security by Design Principles*. URL: https://www.owasp.org/index.php/Security_by_Design_Principles (zitiert auf S. 11).

[Thef] The Open Web Application Security Project (OWASP). *Source Code Analysis Tools*. URL: https://www.owasp.org/index.php/Source_Code_Analysis_Tools (zitiert auf S. 31, 45).

[WAS17] C. Wijayarathna, N. A. G. Arachchilage, J. Slay. „A Generic Cognitive Dimensions Questionnaire to Evaluate the Usability of Security APIs“. In: *Human Aspects of Information Security, Privacy and Trust*. Hrsg. von T. Tryfonas. Cham: Springer International Publishing, 2017, S. 160–173. ISBN: 978-3-319-58460-7 (zitiert auf S. 65).

[Wik] Wikipedia-Webseite. *Java version history*. URL: https://en.wikipedia.org/wiki/Java_version_history (zitiert auf S. 36).

[WT99] A. Whitten, J. D. Tygar. „Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0“. In: *USENIX Security Symposium*. Bd. 348. 1999 (zitiert auf S. 11).

Alle URLs wurden zuletzt am 20.09.2018 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift