Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Erstellung von CryptoExamples in C#

Nico Rusam

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	M. Sc. Kai Mindermann
Beginn am:	22. Mai 2018
Beendet am:	22. November 2018

Kurzfassung

Kontext: Kryptografie und Softwaresicherheit werden in der heutigen Zeit immer wichtiger. Jedoch fehlt Softwareentwicklern häufig das Wissen über diese Themengebiete. Das führt dazu, dass entwickelte Software immer wieder Sicherheitslücken aufweist oder von vornherein nicht sicher ist. Daraus können Angriffe resultieren, was zu einer Gefahr für Länder, Firmen und Privatleute werden kann. Es existieren kryptografische Bibliotheken die entsprechende Algorithmen bereitstellen. Jedoch sind diese häufig schlecht dokumentiert und Beispiele, die zeigen wie die Bibliothek verwendet wird, sind veraltet. Die Probleme mit den veralteten Beispielen reichen von geänderten Schnittstellen bis hin zur Verwendung von nicht mehr sicheren kryptografischen Methoden. *Ziel:* Für die Plattform CryptoExamples werden sichere Codebeispiele für die Sprache C# entwickelt. Zusätzlich werden Richtlinien erarbeitet, die die Anforderungen der Plattform abdecken, sodass mit deren Verwendung die Entwicklung weiterer Beispiele einfacher wird. *Verfahren:* Für verschiedene kryptografische Szenarien werden Codebeispiele erstellt und fortlaufend verbessert. Währenddessen fließen die Erfahrungen, die beim Erstellen der Beispiele gesammelt wurden, in die Erstellung der Richtlinien ein. *Ergebnis:* In dieser Arbeit werden Beispiele für die Sprache C# und die Richtlinien für die Umsetzung weiterer Beispiele mit dieser Sprache erarbeitet. Die Beispiele decken die Szenarien Hashing, digitale Signatur, symmetrische Verschlüsselung sowie asymmetrische Verschlüsselung ab. Des Weiteren werden 25 Richtlinien für die Sprache C# angepasst oder neu erstellt. *Fazit:* Auf der Plattform CryptoExamples wird der Beispielcode bereitgestellt, von dem garantiert wird, dass er sichere kryptografische Verfahren verwendet. Außerdem werden die Beispiele fortlaufend aktualisiert, womit die zukünftige Sicherheit des Codes gewährleistet wird. Zusätzlich gibt es Richtlinien für die Entwicklung neuer Beispiele. Damit soll die Einhaltung der Anforderungen der Plattform sichergestellt werden. Zum jetzigen Zeitpunkt ist die Plattform noch recht klein, sodass die Auswahl an Beispielen zu verschiedenen Programmiersprachen stark eingeschränkt ist.

Inhaltsverzeichnis

1 Einführung **15**
1.1 Motivation 15
1.2 Verwandte Arbeiten 16
1.3 Ziel 17

2 Grundlagen **19**
2.1 Kryptografische Verfahren 19
2.2 Verwendete Technologien 23

3 Anforderungen **25**
3.1 Szenarien 26
3.2 Nutzergruppen 27
3.3 CryptoExamples 27

4 Umsetzungskonzept **33**
4.1 Entwurf 33
4.2 Implementierung 35
4.3 Code Beispiele 37
4.4 Richtlinien 44

5 Zusammenfassung und Ausblick **51**
5.1 Zusammenfassung 51
5.2 Ausblick 51

Literaturverzeichnis **53**

Abbildungsverzeichnis

1.1 Studie über Cyber-Attacken auf 503 mittelständische Unternehmen 16

2.1 Schematische Darstellung der Korrelation der APIs des .Net Frameworks und des .Net Cores 23

3.1 Architektur der CryptoExamples Plattform 28
3.2 Screenshot der Startseite von CryptoExamples 29
3.3 Screenshot der Übersichtsseite für Beispiele in Java 30
3.4 Screenshot eines Beispiels auf CryptoExamples 31

4.1 Darstellung der Zusammenhänge und -arbeit der einzelnen Komponenten des Systems 34
4.2 Schematischer Ablauf der inkrementellen Entwicklung der Beispiele 36

Tabellenverzeichnis

4.1 Auflistung der Codezeilen für mehrere Sprachen über unterschiedliche Beispiele . 44
4.2 Gegenüberstellung von positiven und negativen Effekten auf die Anforderungen . 49

Verzeichnis der Listings

4.1 Die Konfigurationsdatei für Travis CI 36
4.2 Code für das Hashen eines Textes 37
4.3 Der Code für das Signieren und Verifizieren eines Textes 38
4.4 Ver- und Entschlüsselung eines Textes mittels asymmetrischer Verschlüsselung . 39
4.5 Der Code für die Ver- und Entschlüsselung eines Textes mittels Passwort 41

Abkürzungsverzeichnis

AES Advanced Encryption Standard. 19

API Application Programming Interface. 16

CBC Cipher Block Chaining. 20, 35

CCM Counter with CBC-MAC. 35

CI Continuous integration. 24

CTR Counter. 20

CTS Ciphertext Stealing. 35

DES Data Encryption Standard. 19

DLIES Discrete Logarithm Integrated Encryption Scheme. 20

DSA Digital Signature Algorithm. 22

ECB Electronic Code Book. 35

ECIES Elliptic Curve Integrated Encryption Scheme. 20

GCM Galois/Counter-Mode. 20

MD5 Message-Digest Algorithm 5. 21

PBKDF2 Password-Based Key Derivation Function 2. 21

RSA Rivest–Shamir–Adleman. 20

SHA Secure Hash Algorithm. 21

SSL Secure Sockets Layer. 16

TLS Transport Layer Security. 16

1 Einführung

In unserer hochvernetzten Welt gibt es einen ständigen Austausch von Daten. Um diese zu verarbeiten wird entsprechende Software benötigt, die neu geschrieben oder weiterentwickelt werden muss. Sind die Programmierer, die für die Entwicklung zuständig sind, nicht geschult in den Themen Kryptografie oder Sicherheit, kann es leicht passieren, dass Sicherheitslücken entstehen. Dies führt zu Problemen für Länder, Unternehmen sowie für Privatpersonen und kann zur Folge haben, dass ganze Existenzen oder Geschäftszweige zerstört werden.

Um den Angriffsvektor der fehlerhaft programmierten Software zu reduzieren, wurde die Plattform CryptoExamples erschaffen. Sie soll zu einer der zentralen Stellen werden, wenn es um sichere kryptografische Codebeispiele geht. Die Codebeispiele sollen dabei die Anforderungen sicher, minimal, vollständig, ausführbar, getestet und dokumentiert erfüllen. Was diese Anforderungen genau bedeuten wird in Kapitel 3 erläutert.

1.1 Motivation

Immer öfter werden Cyber-Angriffe sowohl auf Unternehmen und Länder, als auch auf Privatleute bekannt. Erfolgreiche Angriffe bedeuten häufig horrende wirtschaftliche Schäden. Diese können nicht nur finanzieller Art sein, sondern auch die Reputation des Unternehmens nachhaltig schädigen. So entstand der deutschen Wirtschaft in den Jahren 2016 und 2017 insgesamt ein Schaden in Höhe von 43,4 Milliarden Euro [18b]. Die Abbildung 1.1 zeigt das Ergebnis einer Umfrage unter deutschen mittelständischen Unternehmen. Sie wurden befragt, ob ihr Unternehmen innerhalb der letzten zwei Jahre von Datendiebstahl, Industriespionage oder Sabotage betroffen war. Dabei konnte festgestellt werden, dass insgesamt 68% der Unternehmen von mindestens einem der drei Angriffsarten betroffen waren. Nur 13% der Unternehmen gaben an, nicht betroffen gewesen zu sein.

Obwohl die größte Gefahr von unbedarften oder leichtgläubigen Nutzern ausgeht (Phishing, Malware, usw.), so ist Software, die aufgrund falscher Programmierung unsicher ist, dennoch Teil des Problems. Grundsätzlich gilt es stets sämtliche Fehlerquellen im Voraus auszumerzen.

So musste der Sicherheitsanbieter Symantec (bekannt durch die Antiviren-Lösung Norton) seinen Geschäftszweig der Zertifikate aufgeben. Dies ist zwar genau betrachtet kein Resultat eines Angriffs auf Symantec, dennoch ist ein solches Verhalten auch durch fehlerhafte Implementierung eines kryptografischen Verfahrens möglich.

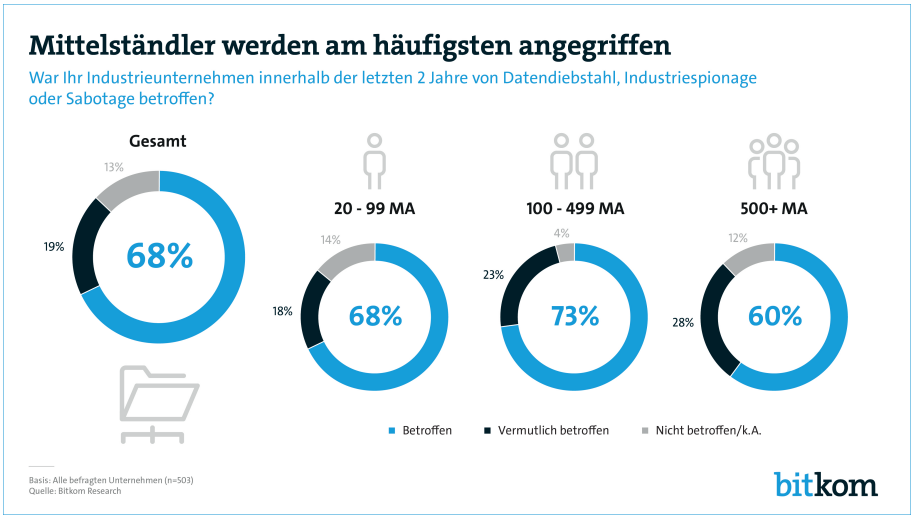

Abbildung 1.1: Studie über Cyber-Attacken auf 503 mittelständische Unternehmen (von bitkom, 2018) [18b]

1.2 Verwandte Arbeiten

Wie und ob die Plattform CryptoExamples Auswirkungen auf die Qualität der Programmierung hat, wurde bereits in einer anderen Arbeit [MW18] untersucht und man kam zu dem Ergebnis, dass Probanden, welche die Plattform verwendet haben um 73% effektiver waren und mögliche Sicherheitslücken um 66% verringert werden konnten.

Neben dieser Arbeit entstehen zeitgleich noch drei weitere Arbeiten, die sich mit der Plattform CryptoExamples befassen. Eine der drei beschäftigt sich im Allgemeinen mit dem Erstellen von Richtlinien und damit, welche Eigenschaften bestimmte Szenarien erfüllen müssen, sodass diese verwendet werden können [Tei18]. Die anderen beiden sind, wie diese Arbeit, auf eine bestimmte Sprache festgelegt. Diese Sprachen sind JavaScript [Hir18] und Python [Klo18].

Georgiev et al. [GIJ+12] stellten fest, dass

> "SSL[1] certifcate validation is completely broken in many critical software applications and libraries. […] The root cause of most of these vulnerabilities is the terrible design of the APIs to the underlying SSL libraries."

Somit wurde gezeigt, dass der Hauptgrund für Schwachstellen im schlechten Design der Application Programming Interfaces (APIs) liegen.

In der Arbeit von Green und Smith [GS16] wird auf die Notwendigkeit eingegangen, dass kryptografische APIs einfach benutzbar sein müssen. Entwickler, die eine kryptografische Bibliothek verwenden, sollen sich nicht in eine solche Bibliothek und das zugrundeliegende kryptografische System einarbeiten müssen. Zur Stärkung der Sicherheit über alle Systeme hinweg sollen sich Sicherheitsexperten auf entwicklerfreundliche und entwicklerorientierte Ansätze konzentrieren.

[1] Secure Sockets Layer (SSL), mittlerweile umbenannt in Transport Layer Security (TLS), ist ein Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet.

1.3 Ziel

Diese Arbeit behandelt die Erstellung von Beispielen für die CryptoExamples Plattform in der Programmiersprache C# sowie die daraus entstandenen Richtlinien. Jene sollen bei der Erstellung weiterer Beispiele, speziell für die Sprache C#, verwendet werden. So wird sichergestellt, dass der Code einheitlich aufgebaut ist und die Anforderungen der Plattform CryptoExamples erfüllt sind. Des Weiteren wird auf Schwierigkeiten eingegangen, die beim Erstellen der Beispiele auftraten sowie ein Ausblick auf mögliche zukünftige Arbeiten gegeben.

2 Grundlagen

2.1 Kryptografische Verfahren

Die Kryptografie (aus dem Griechischen von kryptós = verborgen und gráphein = schreiben) beschäftigt sich mit Verfahren, die einen Klartext (engl. Plaintext) in einen möglichst unlesbaren Schlüsseltext (auch als Chiffretext oder Chiffrat, engl. Ciphertext) überführen [KW11].

Neben der *Vertraulichkeit*, die durch das Verschlüsseln erreicht wird, ist auch die *Integrität* ein wichtiger Bestandteil von kryptografischen Verfahren. Dadurch kann garantiert werden, dass Daten nicht von Unbefugten verändert werden. Neben den beiden Sicherheitszielen *Vertraulichkeit* und *Integrität* gibt es noch die *Authentizität*. Diese sichert zu, dass der tatsächliche Ursprung von Daten klar identifizierbar beziehungsweise nachvollziehbar ist. Zur Veranschaulichung ein Beispiel: Alice schickt Bob eine Nachricht. Für Bob stellt sich die Frage, ob diese tatsächlich von Alice kommt oder von einer (unbekannten) dritten Partei [KW11].

Zur Erreichung dieser drei Sicherheitsziele gibt es verschiedene Verfahren, die alle ihre Vor- und Nachteile haben. Im Folgenden wird eine Auswahl der wichtigsten Konzepte kurz vorgestellt.

2.1.1 Symmetrische Verschlüsselungsverfahren

Bei symmetrischen Verschlüsselungsverfahren wird mit demselben Schlüssel verschlüsselt und entschlüsselt. Sie sollen die Vertraulichkeit von Daten gewährleisten.

Beispiele für symmetrische Verschlüsselungsalgorithmen:

- Advanced Encryption Standard (AES) ist eine Blockchiffre. Das ist ein Algorithmus der einen Klartext fester Bitlänge (z. B. 128 Bit) in einen Chiffretext mit der gleichen Bitlänge umwandelt. Dazu wird ein Schlüssel verwendet. Sollen Klartexte mit einer anderen Länge verschlüsselt werden, müssen sogenannte Betriebsarten eingesetzt werden. Je nach Betriebsart können neben der Vertraulichkeit noch weitere Sicherheitsziele zugesichert werden [18f].
- Data Encryption Standard (DES)
- Blowfish

Vorteile:

- Effizienter als asymmetrischen Verfahren.

Nachteile:

- Da das Verfahren auf den gleichen Schlüssel für die Ver- und Entschlüsselung setzt, muss ein Austausch zwischen Sender und Empfänger über den Schlüssel stattfinden. Dies ist kein triviales Problem. Sollte der Schlüssel abgefangen werden (z. B. durch einen Man-in-the-middle) ist die Vertraulichkeit der Daten nicht mehr gewährleistet.

2.1.2 Betriebsarten

Im Gegensatz zu Blockchiffren, die nur auf die Verschlüsselung von Klartexten mit einer festen Länge ausgelegt sind, dienen Betriebsarten der Verschlüsselung von Texten (nahezu) beliebiger Länge. Des Weiteren können die Betriebsarten auch zu einer stärkeren Verschlüsselung führen, als die zugrundeliegende Blockchiffre für sich genommen [18f].

Beispiele für empfohlene Betriebsarten:

- Galois/Counter-Mode (GCM) ist ein Modus, der zusätzlich die Authentizität der Daten gewährleistet [18f].
- Cipher Block Chaining (CBC)
- Counter (CTR)

2.1.3 Asymmetrische Verschlüsselungsverfahren

Für die Ver- und Entschlüsselung werden zwei verschiedene Schlüssel benutzt. Die Schlüssel werden vom Empfänger der Nachricht erstellt. Dabei muss einer der Schlüssel geheim gehalten werden, während der andere veröffentlicht wird (z. B. auf der Homepage). Soll ein geheimer Text versendet werden, wird der öffentliche Schlüssel benutzt, um den Klartext zu verschlüsseln. Dieser Chiffretext kann nur mithilfe des privaten Schlüssels entschlüsselt werden.

Beispiele für asymmetrische Verschlüsselungsverfahren:

- Rivest–Shamir–Adleman (RSA) ist im PKCS #1: RSA Cryptography Specifications Version 2.2 spezifiziert[1].
- Elliptic Curve Integrated Encryption Scheme (ECIES)
- Discrete Logarithm Integrated Encryption Scheme (DLIES)

Vorteile:

- Solange der geheime Schlüssel geheim bleibt und eine ausreichende Schlüssellänge verwendet wurde, ist die Verschlüsselung praktisch nicht zu brechen.
- Der Schlüsselaustausch ist aufgrund der Öffentlichkeit des öffentlichen Schlüssels kein Problem.

[1] RSA Spezifikation https://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf (Stand 20.11.2018)

Nachteile:

- Ineffizient im Vergleich zu symmetrischen Verfahren.
- Für eine bidirektionale Kommunikation müssen beide Seiten einen öffentlichen Schlüssel zur Verfügung stellen.

2.1.4 Hashing

Algorithmen, die einen Bitstring beliebiger Länge auf einen Bitstring fester Länge übertragen, werden als Hashing bezeichnet. Der Wert des Bitstrings fester Länge wird Hashwert genannt. Dabei muss gewährleistet sein, dass aus einem gehashten Wert nicht der ursprüngliche Wert extrahiert bzw. abgeleitet werden kann. Eine Hashfunktion gilt als kollisionsresistent, wenn es praktisch unmöglich ist, dass zwei verschiedene Bitstrings denselben Hashwert ergeben [18f].

Ein häufiges Einsatzgebiet ist das Speichern von Passwörtern. So wird nur der Hashwert eines Passworts gespeichert und nicht das Passwort im Klartext. Aufgrund der Kollisionsresistenz kann bei der Prüfung der Passworteingabe vom eingegebenen Passwort der Hashwert gebildet werden. Dieser wird mit dem gespeicherten Hashwert abgeglichen.

Ein weiteres Einsatzgebiet sind Prüfsummen. Hierfür wird z. B. von einer Datei der Hashwert gebildet und dieser veröffentlicht. Soll diese Datei heruntergeladen und sichergegangen werden, dass die heruntergeladene Datei mit jener auf der Webseite übereinstimmt, so kann von dieser Datei ebenfalls der Hashwert gebildet werden. Er wird anschließend mit dem auf der Webseite abgeglichen. Stimmen diese überein handelt es sich um dieselbe Datei.

Beispiele für Hashverfahren:

- Secure Hash Algorithm (SHA) gehört zur Gruppe der kryptologischen Hashfunktionen und gilt damit als kollisionsresistent. SHA gibt es in mehreren Ausführungen, von denen die beiden Versionen SHA2 und SHA3 aktuell als sicher gelten (davon ausgenommen ist SHA-224, der zu SHA2 gehört) [18f].
- Password-Based Key Derivation Function 2 (PBKDF2) gehört zur Gruppe der Passwort-Hashfunktionen und wird dazu verwendet, aus einem Passwort einen Schlüssel abzuleiten. Dieser kann wiederum in einem symmetrischen Verschlüsselungsverfahren verwendet werden.
- Message-Digest Algorithm 5 (MD5) wurde früher häufig verwendet, gilt jedoch mittlerweile als nicht mehr sicher.

Vorteile:

- Aus einem Hashwert können die zugrundeliegenden Daten nicht abgeleitet werden.
- Sehr einfach zu verwenden.

Nachteile:

- Sollte die zugrundeliegende Hashfunktion unsicher werden (so geschehen bei MD5), müssen alle damit erzeugten Hashwerte ersetzt werden, da ansonsten die Sicherheit nicht mehr gewährleistet werden kann.

- Die Kollisionsresistenz stellt ein Sicherheitsrisiko dar. Sogenannte Wörterbuch-Angriffe können die Folge sein. Dabei wird von häufig benutzten Werten (z. B. bei Passwörtern das Wort “passwort“) von vornherein der Hashwert berechnet und dieser mit seinem ursprünglichen Wert im Wörterbuch gespeichert. Somit muss bei einem vorhandenen Hashwert lediglich im Wörterbuch nachgesehen werden, um den zugehörigen Klartext zu erhalten. Um diese Angriffe zu erschweren wird vor dem Hashen häufig ein sogenannter Salt dem Klartext hinzugefügt.

2.1.5 Salt

Ein Salt ist eine zufällige Zeichenfolge, die dem eigentlichen Klartext angehängt wird. Damit werden Probleme wie Wörterbuchangriffe bei Hashing deutlich erschwert.

Vorteile:

- Da viele Verschlüsselungsalgorithmen die Länge des Klartextes nicht verändern, kann ein Salt dazu verwendet werden dies zu tun.
- Erhöht die Sicherheit.

Nachteile:

- Wird ein Salt zufällig gewählt (im Gegensatz zu z. B. dem Benutzernamen als Salt), muss dieser auch gespeichert werden. Das benötigt zusätzlichen Speicherplatz.
- Die Erzeugung und Verwendung eines Salts führt zu einem höheren Rechenaufwand.

2.1.6 Digitale Signatur

Die Urheberschaft und Integrität einer Nachricht werden durch eine digitale Signatur sichergestellt. Dies ist möglich, da eine digitale Signatur ein asymmetrisches Kryptosystem ist. Der Sender bildet zuerst einen Hashwert aus den zu signierenden Daten und anschließend wird mit dem privaten Schlüsselpaar die Signatur erstellt. Der Empfänger der Daten kann mit dem öffentlichen Schlüssel die Authentizität der Daten prüfen.

Beispiele für digitale Signaturalgorithmen:

- RSA ist der gleiche Algorithmus wie bei den asymmetrischen Verschlüsselungsverfahren.
- Digital Signature Algorithm (DSA)
- Merkle-Signaturen

Vorteile:

- Es kann geprüft werden, ob die Daten von dem angegebenen Sender kommen.
- Aufgrund des asymmetrischen Kryptosystems ist ein Schlüsselaustausch nicht nötig.

Nachteile:

- Es ist aufwendig, da zuerst gehashed und danach die Signatur erstellt werden muss.

- Wird immer das gleiche Schlüsselpaar verwendet, so sind alle Signaturen bei Kompromittierung des privaten Schlüssels nicht mehr sicher. Werden auf der anderen Seite mehrere Schlüsselpaare verwendet, so ist die Speicherung und Verwaltung der privaten Schlüssel aufwendiger.

2.2 Verwendete Technologien

2.2.1 C#

C# ist eine typsichere, objektorientierte Programmiersprache, die von Microsoft entwickelt wurde. Sie ist im Jahr 2001 erschienen und stark von den Sprachen Java und C++ beeinflusst. Die aktuelle Version ist 7.3.

Laut dem TIOBE-Index[2] ist C# aktuell (Stand 16.11.2018) auf dem 6. Platz hinter Python (4) und Visual Basic .Net (5) aber vor JavaScript (7) und PHP(8).

2.2.2 .Net Framework und .Net Core

Das .Net Framework ist die Laufzeitumgebung für die Programmiersprachen Visual Basic .Net, F# und C#. Ursprünglich war das .Net Framework ausschließlich auf Microsofts Betriebssystem Windows lauffähig. Dies ist immer noch der Fall. Jedoch gibt es mittlerweile Ableger, wie das plattformunabhängige Open-Source Framework .Net Core. Wie die Abbildung 2.1 zeigt, haben das ursprüngliche .Net Framework und das .Net Core zwar auch gemeinsame APIs, jedoch besitzen beide größtenteils eigene APIs. Für die Bachelorarbeit wird das .Net Core Framework verwendet. Hierfür ist die Plattformunabhängigkeit der entscheidende Faktor.

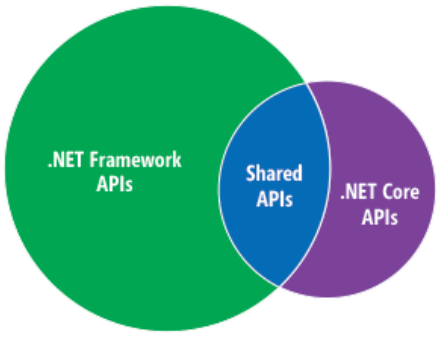

Abbildung 2.1: Schematische Darstellung der Korrelation der APIs des .Net Frameworks und des .Net Cores (von Phillip Carter, Zlatko Knezevic, 2016) [18a]

[2]Eine monatlich aktualisierte Rangliste, die Programmiersprachen nach ihrer Popularität ordnet. https://www.tiobe.com/tiobe-index/ (Stand 16.11.2018)

2.2.3 GitHub

GitHub ist ein Onlinedienst für Versionsverwaltung. Dies geschieht mit dem Verwaltungssystem Git woher sich auch der Name ableitet. Es wurde im Februar 2008 veröffentlicht und am 4. Juni 2018 von Microsoft für 7,5 Milliarden Dollar gekauft [18h]. GitHub hat laut eigenen Angaben (Stand 16.11.2018) über 31 Millionen Nutzerkonten sowie mehr als 2,1 Millionen Organisationen. Die Anzahl der Repositories liegt bei über 96 Millionen [18i].

2.2.4 Travis CI

Travis Continuous integration (CI) ist ein Webdienst für kontinuierliche Integration, der die Erstellung und das Testen von Softwareprojekten auf GitHub ausführt. Er wurde 2011 in Berlin erstellt und 2013 veröffentlicht. Die Konfiguration erfolgt über eine YAML-Datei[3] in der Parameter angegeben werden, die für das Erstellen und Ausführen des Projektes benötigt werden (z. B. die Programmiersprache und die verwendete Laufzeitumgebung).

2.2.5 SonarCloud/SonarQube

SonarCloud ist ein Onlineservice der Firma SonarSource. Er stellt ein Dashboard bereit, auf dem Daten aus der statischen Codeanalyse, grafisch aufbereitet, dargestellt werden. Die Daten werden mit dem SonarQube Scanner erstellt. Bei der Codeanalyse wird unter anderem nach doppeltem Code sowie potenziellen Fehlern gesucht. SonarQube unterstützt mittlerweile mehr als 20 Programmiersprachen.

2.2.6 NuGet

NuGet ist der von Microsoft unterstützte Mechanismus, um Software-Komponenten zu teilen. Dieser definiert außerdem wie Pakete für .Net erstellt, gehostet und verarbeitet werden. Mit anderen Worten ist NuGet der Paketmanager für .Net und kümmert sich unter anderem darum, dass Abhängigkeiten zu anderen Komponenten automatisch geladen werden. Die Pakete sind Zip-Dateien in denen kompilierter Code, zusätzlich benötigte Dateien und ein Manifest, das Beschreibungen des Paketes beinhaltet (wie z. B. die Versionsnummer) enthalten sind.

[3] Ist eine vereinfachte Auszeichnungssprache mit der Dateiendung *.yml

3 Anforderungen

Die Beispiele der CryptoExamples Plattform müssen sechs[1] Anforderungen erfüllen. Diese sind:

- sicher – Die verwendeten Algorithmen und Konzepte müssen zum Zeitpunkt der Erstellung der Beispiele als sicher gelten. Was als sicher gilt, kann der Tabelle auf der CryptoExamples-Guidelines GitHub-Seite entnommen werden[2]. Diese Tabelle wird in regelmäßigen Abständen aktualisiert, wodurch sichergestellt werden soll, dass stets sichere Algorithmen und Konzepte verwendet werden.
- minimal – Die Beispiele sollen möglichst wenig Zeilen Code enthalten. Jedoch muss trotzdem sichergestellt sein, dass sie leicht verständlich und gut lesbar bleiben. Somit ergibt sich ein Konflikt zwischen minimal und dokumentiert. In solch einem Fall sind die Lesbarkeit und Verständlichkeit stets vorzuziehen.
- vollständig – Die Beispiele sollen so aufgebaut sein, dass sie von einem Nutzer kopiert und in ein Projekt eingefügt werden können, ohne dass dabei Probleme beim Ausführen des Codes entstehen.
- ausführbar – Die Beispiele lassen sich automatisiert bauen und ausführen. Dies erfordert, dass mögliche externe Bibliotheken automatisch nachgeladen werden.
- getestet – Die Beispiele lassen sich automatisiert testen und es müssen Tests geschrieben werden, die die Funktionalität der Beispiele überprüfen.
- dokumentiert – Die Beispiele müssen sowohl Klassen- als auch Zeilenkommentare haben. Die Kommentare sollen wichtige und nicht eindeutige Codestellen besser verständlich machen.

[1] Ursprünglich waren sieben Anforderungen vorgegeben. Da allerdings die Anforderungen *vollständig* und *kopierbar* eine nahezu identische Definition besitzen, werden sie in der Anforderung *vollständig* zusammengefasst.

[2] Szenario spezifische Richtlinien https://github.com/cryptoexamples/CryptoExamples-Guidelines#overview-of-secure-cryptographic-algorithms-and-parameter-choices (Stand 01.11.2018)

3.1 Szenarien

Die Szenarien stellen die unterschiedlichen kryptografischen Verfahren dar, zu denen sichere Codebeispiele entwickelt werden sollen. Die Szenarien können Unterpunkte besitzen, wenn es verschiedene Anwendungsfälle für das einzelne Szenario gibt.

1. Symmetrische Verschlüsselung
    a) Textverschlüsselung mit einem Passwort: Ver- und Entschlüsselung eines Textes mit symmetrischer Verschlüsselung. Dabei dient ein textuelles Passwort als Geheimnis. Der ver- und im Anschluss entschlüsselte Text muss mit dem ursprünglichen Text übereinstimmen.
    b) Textverschlüsselung mit einem Schlüssel: Ver- und Entschlüsselung eines Textes mit symmetrischer Verschlüsselung. Dabei dient ein automatisch generierter Schlüssel als Geheimnis. Der ver- und im Anschluss entschlüsselte Text muss mit dem ursprünglichen Text übereinstimmen.
    c) Text in Datei verschlüsseln: Ver- und Entschlüsselung einer Datei mit symmetrischer Verschlüsselung. Dabei wird ein Text mit einem Passwort verschlüsselt und anschließend in einer Datei gespeichert. Für die Entschlüsselung wird die Datei gelesen und danach der Inhalt mit dem Passwort entschlüsselt. Der Text der in verschlüsselter Form in die Datei gespeichert wurde, muss mit dem entschlüsselten Text aus der Datei übereinstimmen.
2. Asymmetrische Verschlüsselung
    a) Textverschlüsselung: Ver- und Entschlüsselung eines Textes mit asymmetrischer Verschlüsselung. Der Text wird dafür mit dem öffentlichen Teil eines neu generierten Schlüsselpaares verschlüsselt und anschließend mit dem privaten Teil wieder entschlüsselt. Der ver- und im Anschluss entschlüsselte Text muss mit dem ursprünglichen Text übereinstimmen.
3. Schlüsselspeicher: Ein Schlüssel wird in einem Schlüsselspeicher gespeichert und anschließend aus dem Speicher heraus verwendet. Der Schlüssel muss vor und nach dem Speichern identisch sein.
4. Hashing
    a) Hashwert aus Text: Von einem Text soll der Hashwert gebildet werden. Der Hashwert eines vorher definierter Textes muss mit einem validierten Ergebnis übereinstimmen. Hierfür muss manuell eine externe Validierung durchgeführt werden (z. B. mit einem online SHA Generator[3]).
5. Digitale Signaturen
    a) Text Signieren: Es soll das Signieren und Verifizieren eines Textes gezeigt werden. Die Verifikation des signierten Textes muss dabei einen akzeptierten Status haben.

[3] Online SHA-Generator https://approsto.com/sha-generator/(Stand 10.11.2018)

3.2 Nutzergruppen

Grundsätzlich gibt es vier verschiedenen Nutzergruppen, die mit oder an CryptoExamples arbeiten. Diese sind:

- Entwickler, die sichere Software schreiben möchten.
- Mitwirkende, die weitere Beispiele für andere Sprachen erstellen oder bestehende Beispiele verbessern bzw. aktualisieren.
- Krypto-Experten, sind Personen, die sich im Bereich der Kryptografie besonders gut auskennen. Sie sind diejenigen, die Änderungen an Beispielen akzeptieren müssen, bevor diese auf der Webseite veröffentlicht werden.
- Projekt-Eigentümer, ist die letzte Kontrollinstanz bevor Beispiele auf der Webseite veröffentlicht werden.

3.3 CryptoExamples

Die Plattform CryptoExamples[4] wurde von Kai Mindermann ins Leben gerufen und soll zu einer der zentralen Stellen werden, wenn es um sichere kryptografische Codebeispiele geht [MW18]. Dabei sollen die Beispiele nach Möglichkeit die Anforderungen *sicher*, *minimal*, *vollständig*, *ausführbar*, *getestet* und *dokumentiert* erfüllen (die Anforderungen sind zu Beginn von Kapitel 3 beschrieben).

3.3.1 Architektur

Abbildung 3.1 zeigt die Architektur von CryptoExamples. In dem Haupt-Repository sind die Daten für die Erstellung der Webseite enthalten. In Neben-Repositorys werden die Beispiele für verschiedene Programmiersprachen entwickelt. Jeder interessierte Entwickler, der bei der Erstellung von Beispielen Mitarbeiten möchte, kann den Fork eines Repositorys abrufen und darin seine Änderungen vornehmen. Anschließend kann er mittels Pull Request seine Änderungen dem Projekt zur Verfügung stellen. Ein Krypto-Experte akzeptiert die Änderungen oder lehnt diese ab. Sollten sie akzeptiert werden, muss der Projekteigentümer als letzte Instanz zustimmen, dass die Beispiele auf der Webseite veröffentlicht werden.

[4] Webseite CryptoExamples https://www.cryptoexamples.com (Stand 10.11.2018)

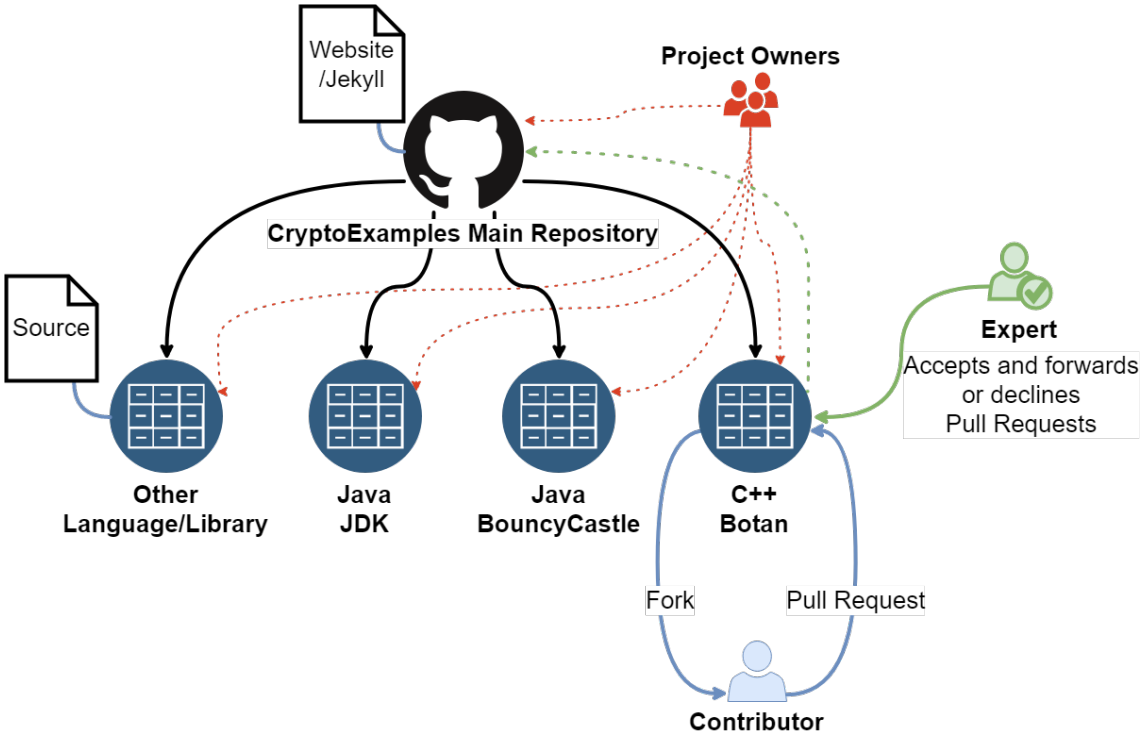

Abbildung 3.1: Architektur der CryptoExamples Plattform (von Kai Mindermann, 2018. MIT-Lizenz) [18g]

3.3.2 Erstellung der Webseite

Die Webseite wird von GitHub mittels GitHub Pages[5] mit den Dateien und Daten des Haupt-Repository von CryptoExamples[6] erstellt. Abbildung 3.2 zeigt die Webseite, die aus dem Haupt-Repository entsteht. Für die Beispiele werden Texte verwendet, die den einzelnen Beispielprojekten als Markdown-Dateien beigelegt sind. In diesen Dateien wird ebenfalls angegeben, wo der Quellcode zu finden ist, der auf der Webseite dargestellt werden soll. Auf diese Weise wird jede Änderung am Code und Text direkt auf der Webseite zur Verfügung gestellt, ohne dass diese angepasst werden müssten. Abbildung 3.4 zeigt die Webseite, wenn ein Beispiel zum Anzeigen ausgewählt wurde. Jedes Beispielprojekt besitzt eine Übersichtsseite, wie in Abbildung 3.3 zu sehen ist. Dort sind die einzelnen Szenarien mit Links zu dem entsprechenden Code aufgelistet.

[5] Webseite von GitHub Pages https://pages.github.com/ (10.11.2018)
[6] Haupt-Repository von CryptoExamples auf GitHub https://github.com/cryptoexamples/CryptoExamples (10.11.2018)

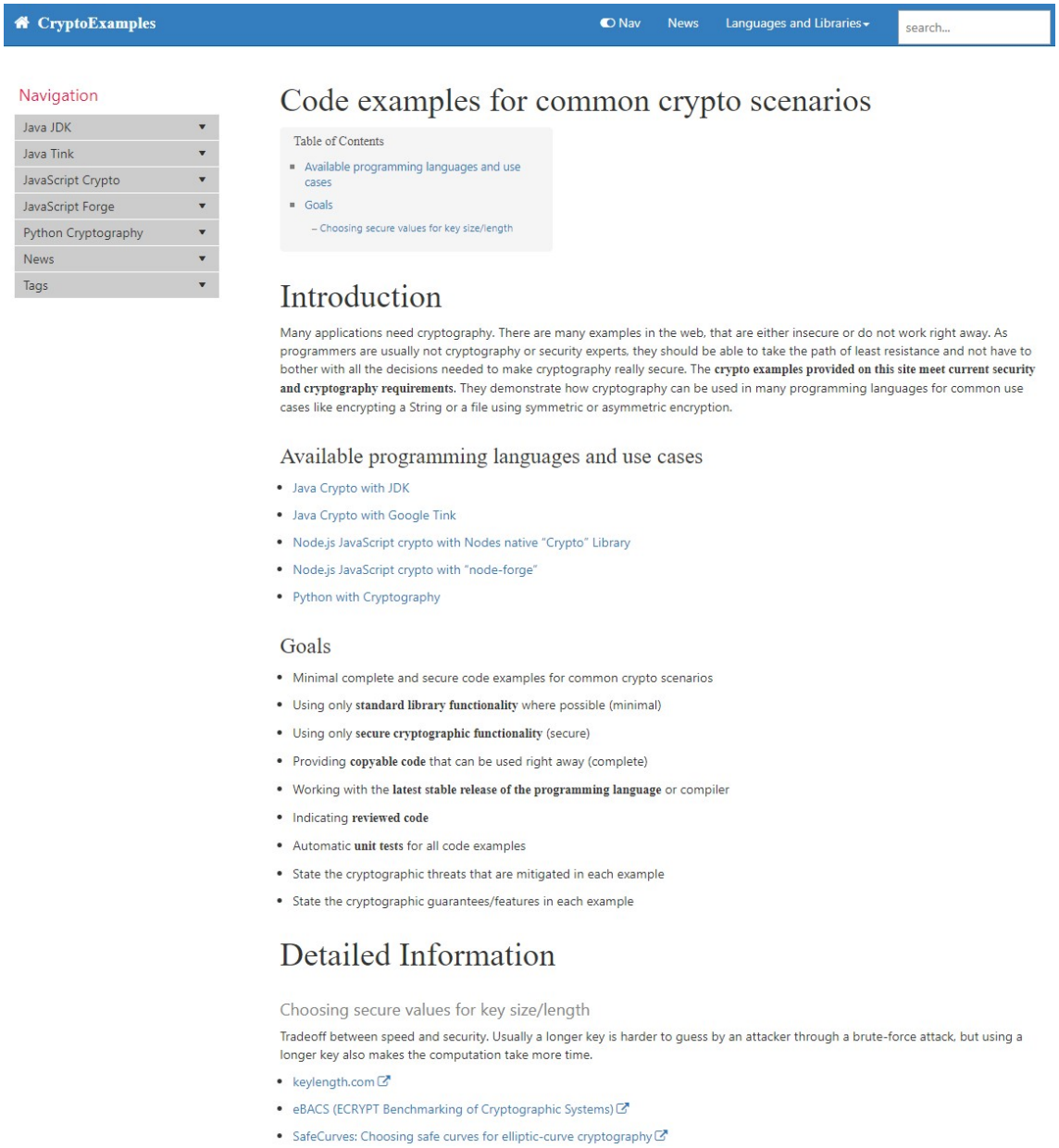

Abbildung 3.2: Screenshot der Startseite von CryptoExamples [18c]

Abbildung 3.3: Screenshot der Übersichtsseite für Beispiele in Java [18d]

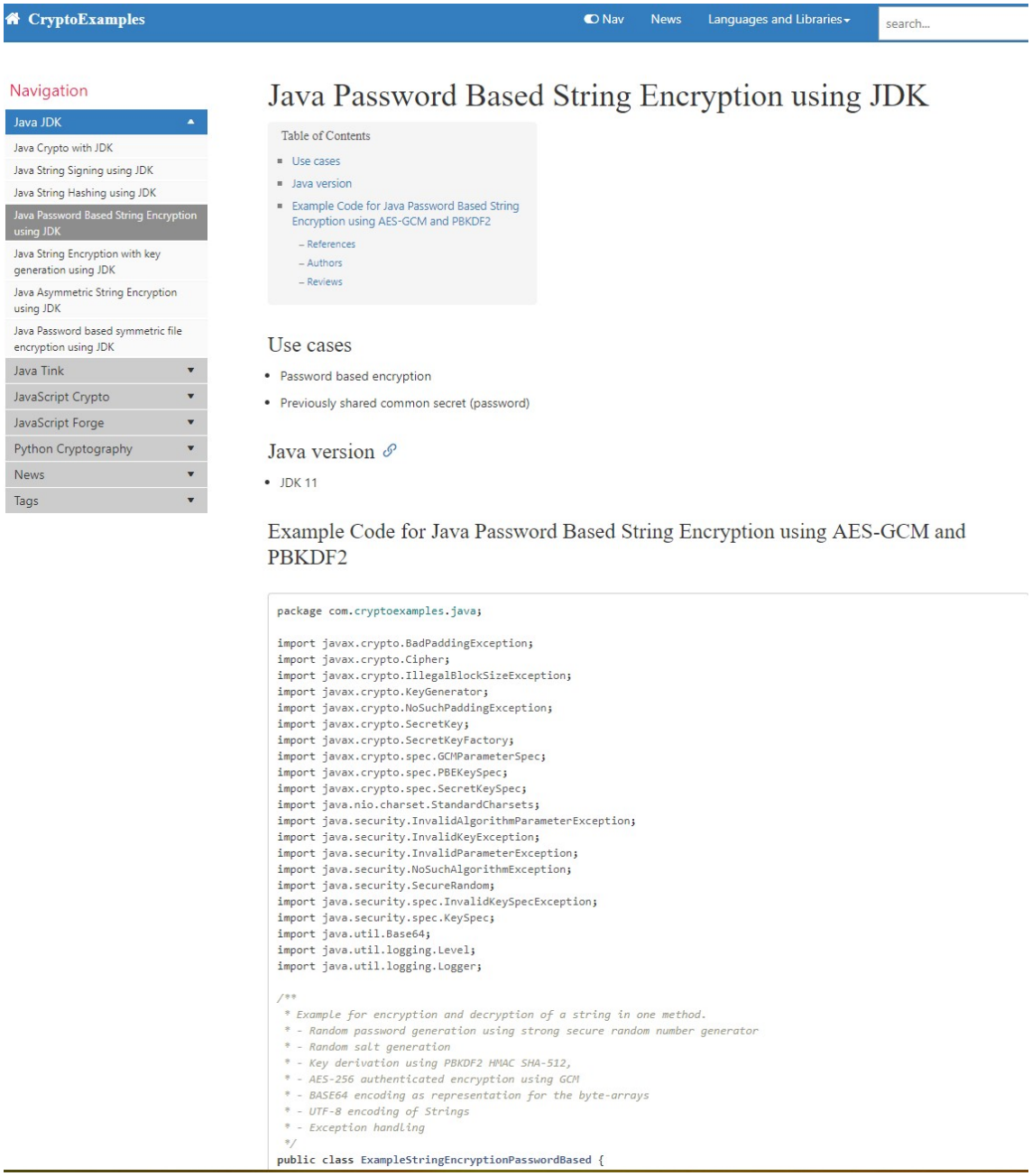

Abbildung 3.4: Screenshot eines Beispiels auf CryptoExamples [18e]

4 Umsetzungskonzept

4.1 Entwurf

Der nachfolgende Entwurf zeigt die beteiligten Komponenten am System und wie diese miteinander interagieren. Des Weiteren wird erläutert, wie sie mit den Anforderungen an die CryptoExamples in Verbindung stehen. Eine grafische Darstellung des Entwurfs ist in Abbildung 4.1 zu sehen.

Für die Quellcodeverwaltung wird die Versionsverwaltung GitHub verwendet, die in Abschnitt 2.2.3 vorgestellt wurde. GitHub wurde aufgrund seiner großen Nutzerschaft ausgewählt, was den Kreis an möglichen Mitwirkenden vergrößert. Sobald ein Mitwirkender seinen Code auf GitHub hochlädt, wird der Dienst für die kontinuierliche Integration Travis CI (beschrieben in Kapitel 2.2.4 und nachfolgend nur noch als Travis bezeichnet) mittels Webhook[1] über den neuen Versionsstand informiert. Travis wird verwendet, um die Anforderungen *ausführbar* und *getestet* abzudecken. Außerdem ist Travis für Open-Source-Projekte kostenlos und unterstützt eine Vielzahl an Programmiersprachen. Nachdem Travis über den neuen Versionsstand informiert wird, lädt es das Repository herunter und beginnt mit dem Erstellungsprozess des Projektes. Dabei werden benötigte Abhängigkeiten mit dem Paketmanager NuGet (siehe Kapitel 2.2.6) aufgelöst. Somit werden Bibliotheken, die nicht zur Standardbibliothek gehören heruntergeladen und dem Projekt zur Verfügung gestellt. Anschließend wird der Code kompiliert.

Nachdem das Projekt erfolgreich erstellt wurde, werden die Softwaretests ausgeführt, die die Funktionalität der Beispiele sicherstellen sollen. Mit dem Erstellen des Projektes wird die Anforderung *ausführbar* abgedeckt und mit den Tests die Anforderung *getestet*. Sollte während des Erstellungsprozesses oder des Ausführens der Tests ein Fehler auftreten und somit das Beispiel nicht ausführbar oder fehlerhaft sein, wird der Mitwirkende, der den Code hochgeladen hat, darüber informiert. Des Weiteren wird auch auf GitHub dargestellt, dass ein Fehler aufgetreten ist. Schlussendlich wird mit dem externen Programm SonnarQube Scanner (aus Abschnitt 2.2.5) die statische Codeanalyse des Quellcodes durchgeführt und deren Ergebnisse werden Sonarcloud bereitgestellt. Dieses bietet seinerseits ein Dashboard mit einer grafisch aufbereiteten Benutzeroberfläche für diese Daten. Die Analyse mit Sonarscanner leistet einen Beitrag dazu, dass die Anforderung der *Sicherheit* erhöht wird. Dabei nicht nur nach Programmierfehlern und sogenannten Code Smells[2], sondern auch nach Sicherheitsverstößen innerhalb des Programmcodes gesucht.

[1] Ein Verfahren zur Kommunikation von Servern, das zum Benachrichtigen bei Ereignissen (wie in diesem Fall, dass ein neuer Code im Repository vorhanden ist) dient.

[2] Code Smells sind bestimmte Strukturen im Programmcode, die auf einen Verstoß gegen grundlegende Konstruktionsprinzipien hinweisen und die Codequalität negativ beeinflussen können.

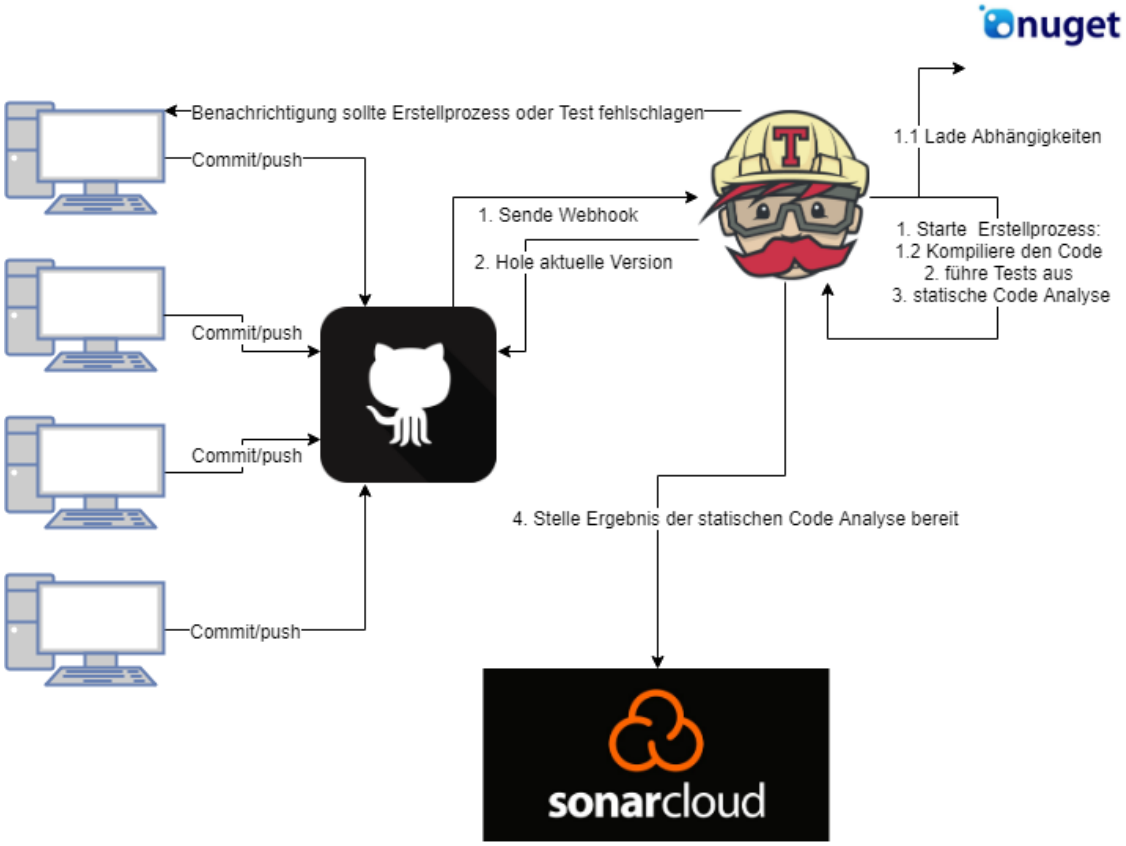

Abbildung 4.1: Darstellung der Zusammenhänge und -arbeit der einzelnen Komponenten des Systems (eigene Grafik)

Dieser Entwurf hat Vorteile gegenüber einem Entwurf ohne automatisiertes Erstellen und Testen. Er macht den Nachteil des zusätzlichen Aufwandes bei der Einrichtung neuer Projekte für Crypto-Examples wieder wett. Zum einen wird durch die Automatisierung verhindert, dass nicht-lauffähiger oder fehlerhafter Code auf der Webseite bereitgestellt wird. Zum anderen besteht der Nachteil des zusätzlichen Aufwandes in der Regel nur zu Beginn des Projektes, da nach korrekter Inbetriebnahme eine Anpassung häufig nicht mehr nötig ist.

Die automatisierte statische Codeanalyse hat ähnliche Vor- und Nachteile wie beim automatisierten Erstellen und Testen. So ist ein weiterer Nachteil, dass die Einrichtung einen Mehraufwand bedeutet. Demgegenüber steht der Vorteil, dass die Mitwirkenden nicht selbst die Analyse starten müssen.

Ein weiterer Vorteil dieses Entwurfes betrifft das separate Laden von Abhängigkeiten. Wären diese im Projekt enthalten, so müsste sich der Mitwirkende stets selbst um die Aktualisierung der Pakete kümmern. Sollte das Aktualisieren vergessen werden, könnte das die Sicherheit und die Lauffähigkeit der Beispiele negativ beeinflussen.

4.2 Implementierung

Für die Implementierung wurde Visual Studio 2017 verwendet, sowie .Net Core in der zu Beginn des Projektes aktuellen Version 2.0. Im Laufe der Arbeit wurde die Version 2.1 veröffentlicht. Da eine der Anforderungen lautet, dass der Code mit der neuesten stabilen Version lauffähig sein soll, wurde auf diese aktualisiert. Zudem wurde versucht sämtliche Beispiele mit der Standardbibliothek (`System.Security.Cryptography`) zu bearbeiten. Dies war jedoch bei zwei Szenarien nicht möglich. Diese sind:

1. Symmetrische Verschlüsselung
2. Schlüsselspeicher

Der Grund dafür ist bei dem Szenario der symmetrischen Verschlüsselung der, dass das Verschlüsselungsverfahren AES angewandt wird. Jedoch unterstützt die Standardbibliothek bisher nur die drei Modi Cipher Block Chaining (CBC), Ciphertext Stealing (CTS) und Electronic Code Book (ECB). Da keiner dieser Modi dem geforderten Counter with CBC-MAC (CCM) oder GCM entspricht, musste eine andere Bibliothek verwendet werden, die diese Möglichkeit bietet. Deshalb wurde die Bibliothek Bouncy Castle[3] verwendet.

Bei dem Szenario Schlüsselspeicher war die Ausgangslage etwas anders. So gibt es in der Standardbibliothek die Möglichkeit einen Schlüsselspeicher zu erstellen. Dieser ist jedoch nur unter Windows verwendbar und verursacht unter anderen Betriebssystemen eine Fehlermeldung. Für die Umsetzung dieses Szenarios muss noch nach einer guten Umsetzungsmöglichkeit gesucht werden.

Für das Erstellen der Beispiele war vorgegeben, dass diese pro Szenario nur aus einer Klasse mit je einer Methode (neben der Main Methode) bestehen sollen in der sämtliche Logik für das gewählte Szenario enthalten ist. Diese Vorgabe hat jedoch für die Anforderungen ein paar Nachteile. So wird auf der einen Seite zwar die Minimalität erhöht, jedoch wird auf der anderen Seite die Vollständigkeit eingeschränkt. Es gibt kaum plausible Einsatzszenarien, bei denen das Ver- und anschließend sofortige Entschlüsseln sinnvoll ist. Die bessere Alternative wäre hier eventuell eine Aufteilung der Szenarien in ihre logischen Teile und die Verwendung einer eigenen Methode für jedes dieser Teile gewesen (z. B. eine Methode für das Verschlüsseln und eine für das Entschlüsseln).

Die Qualität der Beispiele soll, wie in Abbildung 4.2 dargestellt, durch eine inkrementelle Entwicklung[4] sichergestellt werden. So werden anfangs die Beispiele rein auf ihre Funktionalität hin entwickelt und anschließend mit jedem weiteren Iterationsschritt näher an die Anforderungen aus Kapitel 3 herangebracht. Zeitgleich sollen die Erfahrungen, die bei dem Erstellen der Beispiele gemacht werden, für die Erstellung der Guidelines verwendet werden, sodass diese ebenfalls immer besser werden.

Für den Logger wird auf die externe Bibliothek Serilog[5] zurückgegriffen, da dies eine der minimalen Möglichkeiten darstellt einen Logger zu integrieren. Das war notwendig, da der in der Standardbibliothek enthaltene Logger einer aufwändigeren Einrichtung und Initialisierung bedürfe und damit die Anforderung der Minimalität verletzen würde.

[3] Bouncy Castle Website http://bouncycastle.org/csharp/index.html (Stand 01.11.2018)

[4] Ein Vorgehensmodell zur Entwicklung von Software, bei dem zu Beginn sämtliche Anforderungen bekannt sind und daraus eine frühe Version erstellt werden kann, die im Laufe der Zeit immer weiter verbessert wird.

[5] Serilog Webseite https://serilog.net/ (Stand 09.09.2018)

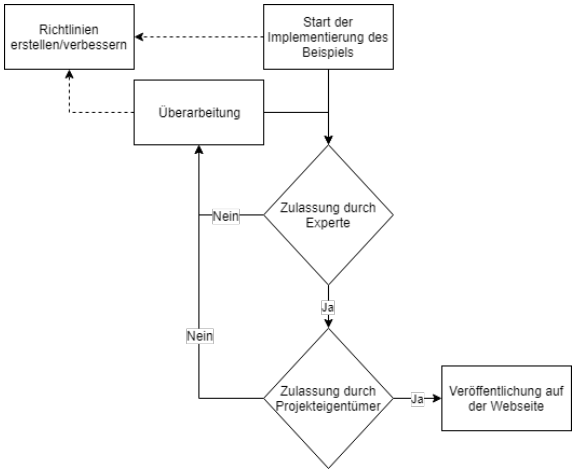

Abbildung 4.2: Schematischer Ablauf der inkrementellen Entwicklung der Beispiele (eigene Grafik)

Für die Konfiguration und Einstellung von Travis wird eine Konfigurationsdatei mit dem Namen *.travis.yml* verwendet, die im Repository des Projektes vorhanden sein muss. Die für dieses Projekt verwendete Datei ist im Quellcode 4.1 zu sehen. Darin enthalten ist die eingesetzte Programmiersprache (Zeile 1), die Version der Laufzeitumgebung (Zeilen 6-9), die Projektdatei (Zeile 2), welche zusätzlichen Erweiterungen installiert werden sollen (Zeilen 3-5) sowie deren Konfiguration (Zeilen 10-19). Zusätzlich sind Informationen enthalten, durch die Travis die Dateien bzw. Ordner zwischenspeichert (Zeilen 20-24). Dadurch muss deren Inhalt nicht jedes Mal neu erstellt werden. Zu beachten ist, dass der Inhalt der Datei von der verwendeten Programmiersprache abhängig ist und somit nicht 1:1 auf andere Projekte übertragen werden kann.

```
1  language: csharp
2  solution: csharp_cryptoexamples.sln
3  install:
4    - dotnet tool install --global dotnet-sonarscanner
5    - dotnet restore
6  matrix:
7    include:
8      - dotnet: 2.1
9        mono: none
10 addons:
11   sonarcloud:
12     organization: "kmindi-github" # the key of the organization
13 before_script:
14   - export PATH="$PATH:$HOME/.dotnet/tools"
15 script:
16   - dotnet sonarscanner begin /k:"csharp-cryptoexamples" /d:sonar.host.url="https://
   sonarcloud.io" /d:sonar.login="$SONAR_TOKEN" /d:sonar.language="cs" /d:sonar.exclusions="
   **/bin/**/*,**/obj/**/*, **/src/ExmapleKeyStorageProvider.cs" /d:sonar.cs.opencover.
   reportsPaths="lcov.opencover.xml" || true
17   - dotnet build csharp_cryptoexamples.sln
18   - dotnet test ./csharp_cryptoexamplesTest/csharp_cryptoexamplesTest.csproj /p:
   CollectCoverage=true /p:CoverletOutputFormat=opencover /p:CoverletOutput=../lcov
19   - dotnet sonarscanner end /d:sonar.login="$SONAR_TOKEN" || true
```

```
20 cache:
21   directories:
22     - '$HOME/.nuget/packages'
23     - '$HOME/.local/share/NuGet/Cache'
24     - '$HOME/.sonar/cache'
```

Quellcode 4.1: Die Konfigurationsdatei für Travis CI

Bei der Implementierung fiel auf, dass die Dokumentation der Standardbibliothek sehr gut und komfortabel ist. So lassen sich für viele Probleme nicht nur Beschreibungen finden, sondern auch Beispielcodes. Nichtsdestotrotz sind die zu findenden Beispiele nicht mit den Anforderungen von CryptoExamples kompatibel da sie z. B. nicht minimal sind und einige zusätzliche Funktionen enthalten. Im Gegensatz dazu ist die externe Bibliothek (Bouncy Castle) fast gar nicht dokumentiert. Beispiele zu finden war ebenso äußerst schwierig und die angebotenen Beispiele waren weit davon entfernt den Anforderungen gerecht zu werden.

4.3 Code Beispiele

4.3.1 Beispielcode Hashing

Im Quellcode 4.2 ist die Implementierung für das Hashing-Szenario zu sehen. Dieser Methode wird ein `string` übergeben, erstellt daraus den Hashwert mit dem SHA512 Algorithmus und gibt den resultierenden Hashwert als `string` zurück. `hashAlgorithm` wurde in Zeile 19 der SHA512-Algorithmus zugewiesen, sodass darauf in Zeile 20 mittels `hashAlgorithm.ComputeHash` der Hashwert für den übergebenen Parameter berechnet wird.

```
1  using System;
2  using System.Security.Cryptography;
3  using System.Text;
4  using Serilog;
5
6  namespace com.cryptoexamples.csharp
7  {
8    /// <summary>
9    /// Example for hashing of a string in one method.
10   /// <para>- SHA-512</para>
11   /// <para>- BASE64 encoding as representation for the byte-arrays</para>
12   /// <para>- UTF-8 encoding of String</para>
13   /// For more information about the used cryptosystem look at: <see href="https://en.
     wikipedia.org/wiki/SHA-2" />
14   /// </summary>
15   public static class ExampleHashing
16   {
17     public static string DemonstrateHashing(string plainText)
18     {
19       HashAlgorithm hashAlgorithm = SHA512.Create();
20       string hashString = Convert.ToBase64String(hashAlgorithm.ComputeHash(Encoding.UTF8.
     GetBytes(plainText)));
21       Log.Information("The hashed value is: {0}", hashString);
```

```
22        return hashString;
23      }
24
25      public static void Main()
26      {
27        Log.Logger = new LoggerConfiguration().WriteTo.Console().CreateLogger();
28        DemonstrateHashing("Text that should be authenticated by comparing the hash of it!");
29      }
30    }
31  }
```

Quellcode 4.2: Code für das Hashen eines Textes

4.3.2 Beispielcode Signing

Der Quellcode 4.3 zeigt die Klasse für das Szenairo des Signierens von Text. Der Eingabeparameter der Demonstrate-Methode ist ein `string` der signiert werden soll. Dafür wird dieser zuerst in seine Byte-Repräsentation konvertiert und schließlich mit dem `rSACryptoServiceProvider` und SHA-512 (`SHA512CryptoServiceProvider`) signiert. Die Signatur wird anschließend wieder in einen `string` konvertiert. Das Verifizieren der Signatur erfolgt analog. Der signierte `string` wird hierbei in seine Byte-Repräsentation konvertiert und anschließend mit dem `rSACryptoServiceProvider` verifiziert.

```
1  using System;
2  using System.Security.Cryptography;
3  using System.Text;
4  using Serilog;
5
6  namespace com.cryptoexamples.csharp
7  {
8    /// <summary>
9    /// Example for cryptographic signing of a string in one method.
10   /// <para>- Generation of public and private RSA 4096 bit keypair</para>
11   /// <para>- SHA-512 with RSA</para>
12   /// <para>- UTF-8 encoding of Strings</para>
13   /// For more information about the used cryptosystem look at: <see href="https://en.
     wikipedia.org/wiki/RSA_(cryptosystem)" />
14   /// </summary>
15   public static class ExampleSignature
16   {
17     public static bool DemonstrateSigning(string plainText)
18     {
19       bool veryfied = false;
20       try
21       {
22         #region SIGNING
23         byte[] originalData = Encoding.UTF8.GetBytes(plainText);
24         //Generate a new key-pair.
25         RSACryptoServiceProvider rSACryptoServiceProvider = new RSACryptoServiceProvider
26         {
27           KeySize = 4096
28         };
```

```
29          byte[] signedData = rSACryptoServiceProvider.SignData(originalData, new
    SHA512CryptoServiceProvider());
30          string signature = Convert.ToBase64String(signedData);
31          #endregion
32
33          #region VERIFIACTION
34          signedData = Convert.FromBase64String(signature);
35          veryfied = rSACryptoServiceProvider.VerifyData(originalData, new
    SHA512CryptoServiceProvider(), signedData);
36          Log.Information("Signature is correct: {0}", veryfied);
37          #endregion
38        }
39        catch (CryptographicException e)
40        {
41          Log.Error("Error: {0}", e.Message);
42        }
43        return veryfied;
44      }
45
46      public static void Main()
47      {
48        Log.Logger = new LoggerConfiguration().WriteTo.Console().CreateLogger();
49        DemonstrateSigning("Text that should be signed to prevent unknown tampering with its
    content.");
50      }
51    }
52  }
```

Quellcode 4.3: Der Code für das Signieren und Verifizieren eines Textes

4.3.3 Beispielcode asymmetrische Verschlüsselung

Das Szenario der asymmetrischen Verschlüsselung eines Textes wird mit dem Quellcode 4.4 gezeigt. Wie bei den vorherigen Beispielen wird der Demonstrate-Methode ein `string` übergeben, der diesmal den Text, der verschlüsselt werden soll, enthält. Der `RSACryptoServiceProvider` erstellt ein neues Schlüsselpaar für die Ver- und Entschlüsselung. Dabei wird eine Schlüsselgröße von 4096 verwendet. Mit den Methoden `SignData` (Zeile 29) und `VerifyData` (Zeile 35) wird schließlich die Signierung und Verifizierung der Daten vorgenommen.

```
1   using System;
2   using System.Security.Cryptography;
3   using Serilog;
4
5   namespace com.cryptoexamples.csharp
6   {
7     /// <summary>
8     /// Example for asymmetric encryption and decryption of a string in one method.
9     /// <para>- Generation of public and private RSA 4096 bit keypair</para>
10    /// <para>- BASE64 encoding as representation for the byte-arrays</para>
11    /// <para>- UTF-8 encoding of Strings</para>
```

```
12    /// For more information about the used cryptosystem look at: <see href="https://en.
    wikipedia.org/wiki/RSA_(cryptosystem)" />
13    /// </summary>
14    public static class ExampleAsymetricStringEncryption
15    {
16      public static string DemonstrateAsymetricStringEncryption(string plainText)
17      {
18        byte[] dataForEncryption = System.Text.Encoding.UTF8.GetBytes(plainText);
19        // Create a new key pair
20        RSACryptoServiceProvider rSACryptoServiceProvider = new RSACryptoServiceProvider
21        {
22          KeySize = 4096
23        };
24        // ENCRYPTION
25        byte[] encryptedData = rSACryptoServiceProvider.Encrypt(dataForEncryption, false);
26        string encryptedString = Convert.ToBase64String(encryptedData);
27
28        // DECRYPTION
29        byte[] decryptedData = Convert.FromBase64String(encryptedString);
30        string decryptedString = System.Text.Encoding.UTF8.GetString(rSACryptoServiceProvider
    .Decrypt(decryptedData, false));
31        Log.Information("Decrypted and original plain text are the same: {0}",
    decryptedString.Equals(plainText));
32        return decryptedString;
33      }
34
35      public static void Main()
36      {
37        Log.Logger = new LoggerConfiguration().WriteTo.Console().CreateLogger();
38        DemonstrateAsymetricStringEncryption("Text that is going to be sent over an insecure
    channel and must be encrypted at all costs!");
39      }
40    }
41  }
```

Quellcode 4.4: Ver- und Entschlüsselung eines Textes mittels asymmetrischer Verschlüsselung

4.3.4 Beispielcode symmetrische Verschlüsselung

Der Quellcode 4.5 für die symmetrische Verschlüsselung mit einem Passwort steht exemplarisch für alle Szenarien der symmetrischen Verschlüsselung. Diese unterscheiden sich nur marginal voneinander, wodurch das mehrfache Darstellen keinen Mehrwert bieten würde. Außerdem sind diese Beispiele die einzigen, bei denen für die Kryptografie auf eine externe Bibliothek zurückgegriffen werden muss. Wie bereits im Kapitel 4.2 beschrieben, liegt das einzig an der Vorgabe des zu verwendenden Modus. Der Demonstrate-Methode wird als `string` sowohl der Klartext als auch das zu verwendende Passwort übergeben. Dem erstellten `pkcs5S2ParametersGenerator` werden in Zeile 34 mit dem Passwort, einem in Zeile 33 zufällig generierten Salt der Länge 16 Bit und einer Zahl (hier 10000) für den Iterationszähler, drei Parameter übergeben. Der Iterationszähler gibt an, wie häufig eine Hashfunktion auf das Passwort angewendet werden soll (dies geschieht in der externen Bibliothek). Der Code zwischen Zeile 35 und 40 dient der Konfiguration des Kryptosystems. Dort

wird z. B. auch der für dieses Szenario vorgeschlagene GCM-Modus (Zeile 38) initialisiert. In den Zeile 43 bis 45 wird schließlich der Klartext in Byte-Repräsentation (Umwandlung erfolgte bereits in Zeile 30) mit einem Authentizitäts-Tag (Zeile 43) versehen und anschließend in Zeile 44 und 45 verschlüsselt. Von Zeile 49 bis 58 erfolgt das Zusammensetzen des Salts, der Nonce[6] und des eigentlichen verschlüsselten Textes mit anschließender Speicherung in einer Byte-Array. In Zeile 59 wird schließlich das zuvor gespeicherte Array wieder in einen `string` konvertiert. Ab Zeile 62 erfolgt die Entschlüsselung. Dazu wird zuerst der `string` wieder in seine Byte-Repräsentation umgewandelt (Zeile 63) und in dem Array `encryptedMessageAsByteArray` gespeichert. Die Zeilen 66 bis 72 werden von dem `encryptedMessageAsByteArray`-Array das Salt und die Nonce extrahiert und anschließend die Konfiguration des Kryptosystems vorgenommen. Anschließend wird der verbleibende Inhalt des Arrays in dem Array `cipherTextAsByteArray` zwischengespeichert. Zwischen Zeile 78 und 83 erfolgt schlussendlich die Prüfung der Authentizität mittels eines `try- catch` Blocks, der bei falscher Authentisierung eine `InvalidCipherTextException` abfängt. Zeile 81 schließt die Entschlüsselung ab und im `return` der Methode wird schlussendlich das entschlüsselte Byte-Array in den Plaintext konvertiert.

```
1  using System;
2  using System.IO;
3  using System.Text;
4  using Org.BouncyCastle.Crypto;
5  using Org.BouncyCastle.Crypto.Engines;
6  using Org.BouncyCastle.Crypto.Generators;
7  using Org.BouncyCastle.Crypto.Modes;
8  using Org.BouncyCastle.Crypto.Parameters;
9  using Org.BouncyCastle.Security;
10 using Serilog;
11
12 namespace com.cryptoexamples.csharp
13 {
14   /// <summary>
15   /// Example for encryption and decryption of a string in one method.
16   /// <para>- AES-256</para>
17   /// <para>- BASE64 encoding as representation for the byte-arrays</para>
18   /// <para>- Random salt generation</para>
19   /// For more information about the used cryptosystem look at: <see href="https://en.
   wikipedia.org/wiki/Advanced_Encryption_Standard" />
20   /// </summary>
21   public static class ExampleStringEncryptionPasswordBased
22   {
23     public static string DemonstrateStringEncryptionPasswordBased(string plainText, string
   password)
24     {
25       //If the password from the user can't be empty or null. Instead this would be the
   same as 'StringEncryptionKeyBased'.
26       if (!string.IsNullOrEmpty(password))
27       {
28         #region ENCRYPTION
29         SecureRandom Random = new SecureRandom();
30         byte[] dataForEncryption = Encoding.UTF8.GetBytes(plainText);
```

[6]Eine Zahl, die nur einmal verwendet werden sollte. Der Name steht für *number used once*.

```
31          Pkcs5S2ParametersGenerator pkcs5S2ParametersGenerator = new
Pkcs5S2ParametersGenerator();
32          byte[] salt = new byte[16];
33          Random.NextBytes(salt);
34          pkcs5S2ParametersGenerator.Init(PbeParametersGenerator.Pkcs5PasswordToBytes(
password.ToCharArray()), salt, 10000);
35          KeyParameter key = (KeyParameter)pkcs5S2ParametersGenerator.
GenerateDerivedMacParameters(256);
36          byte[] nonce = new byte[16];
37          Random.NextBytes(nonce);
38          GcmBlockCipher gcmBlockCipher = new GcmBlockCipher(new AesEngine());
39          AeadParameters aeadParameters = new AeadParameters(new KeyParameter(key.GetKey()),
128, nonce, salt);
40          gcmBlockCipher.Init(true, aeadParameters);
41
42          // Generate ciphertext with authentication tag.
43          byte[] cipherTextAsByteArray = new byte[gcmBlockCipher.GetOutputSize(
dataForEncryption.Length)];
44          int length = gcmBlockCipher.ProcessBytes(dataForEncryption, 0, dataForEncryption.
Length, cipherTextAsByteArray, 0);
45          gcmBlockCipher.DoFinal(cipherTextAsByteArray, length);
46          byte[] cipherTextBytes;
47
48          //PREPED SALT AND NONCE
49          using(MemoryStream memoryStream = new MemoryStream())
50          {
51            using(BinaryWriter binaryWriter = new BinaryWriter(memoryStream))
52            {
53              binaryWriter.Write(salt);
54              binaryWriter.Write(nonce);
55              binaryWriter.Write(cipherTextAsByteArray);
56            }
57            cipherTextBytes = memoryStream.ToArray();
58          }
59          string cipherText = Convert.ToBase64String(cipherTextBytes);
60          #endregion
61
62          #region DECRYPTION
63          byte[] encryptedMessageAsByteArray = Convert.FromBase64String(cipherText);
64          salt = new byte[16];
65
66          using(BinaryReader binaryReader = new BinaryReader(new MemoryStream(
encryptedMessageAsByteArray)))
67          {
68            salt = binaryReader.ReadBytes(salt.Length);
69            nonce = binaryReader.ReadBytes(16);
70            gcmBlockCipher = new GcmBlockCipher(new AesEngine());
71            aeadParameters = new AeadParameters(new KeyParameter(key.GetKey()), 128, nonce,
salt);
72            gcmBlockCipher.Init(false, aeadParameters);
73
74            cipherTextAsByteArray = binaryReader.ReadBytes(encryptedMessageAsByteArray.Length
 - salt.Length - nonce.Length);
```

```
75            byte[] decryptedTextAsByteArray = new byte[gcmBlockCipher.GetOutputSize(
    cipherTextAsByteArray.Length)];
76
77            // CHECK AUTHENTICATION
78            try
79            {
80              length = gcmBlockCipher.ProcessBytes(cipherTextAsByteArray, 0,
    cipherTextAsByteArray.Length, decryptedTextAsByteArray, 0);
81              gcmBlockCipher.DoFinal(decryptedTextAsByteArray, length);
82
83            }
84            catch (InvalidCipherTextException e)
85            {
86              Log.Error("Error: {0}", e.Message);
87              return null;
88            }
89            Log.Information("Decrypted and original plain text are the same: {0}", plainText.
    Equals(Encoding.UTF8.GetString(decryptedTextAsByteArray)));
90            return Encoding.UTF8.GetString(decryptedTextAsByteArray);
91          }
92          #endregion
93        }
94        else
95        {
96          Log.Error("Error: {0}", "Password can't be empty or null!");
97          return null;
98        }
99      }
100
101     public static void Main()
102     {
103       Log.Logger = new LoggerConfiguration().WriteTo.Console().CreateLogger();
104       DemonstrateStringEncryptionPasswordBased("Text that is going to be sent over an
    insecure channel and must be encrypted at all costs!", "SuperSafe");
105     }
106   }
107 }
```

Quellcode 4.5: Der Code für die Ver- und Entschlüsselung eines Textes mittels Passwort

Szenario	C#	Java	JavaScript	Python
Symmetrische Verschlüsselung mit Passwort	84	59	43	50
Asymmetrische Verschlüsselung	28	35	50	43
Hashing	22	25	33	22
Digitale Signatur	37	37	48	46

Tabelle 4.1: Auflistung der Codezeilen für mehrere Sprachen über unterschiedliche Beispiele

4.3.5 Auffälligkeiten der Beispiele

Bei den Beispielen fällt auf, dass diejenigen, die auf die Standardbibliothek setzen, deutlich kompakter ausfallen als das Beispiel, das auf eine externe Bibliothek zurückgreift. Das liegt vor allem daran, dass bei der externen Bibliothek die Konfiguration selbst vorgenommen werden muss. Diese Konfiguration ist dazu noch sehr komplex und benötigt einige Zeit der Einarbeitung. Das verhält sich mit der Standardbibliothek anders. Dort fällt die Verwendung der kryptografischen Systeme deutlich leichter, da nicht alle Einstellungen vorgenommen werden müssen.

4.3.6 Vergleich des Codes mit Beispielen anderer Sprachen

Zum jetzigen Zeitpunkt sind Beispiele für die Sprachen Java, JavaScript und Python auf CryptoExamples vorhanden. Um die für C# erstellten Beispiele mit den für andere Sprachen erstellten zu vergleichen, wurde die Anzahl der Codezeilen der einzelnen Szenarien gegenübergestellt und in Tabelle 4.1 aufgelistet. Die Zahlen stammen von Sonarcloud aus der Analyse der jeweiligen Beispiele. Für die symmetrische Verschlüsselung wurde nur das Szenario der Textverschlüsselung mit einem Passwort betrachtet. Dabei fällt auf, dass C# in allen Szenarien, in denen die Standardbibliothek verwendet wurde, die wenigsten Zeilen benötigt. Nur bei der digitalen Signatur ist Java und beim Hashing Python gleichauf. Bei dem Szenario, in dem eine externe Bibliothek zum Einsatz kommt (symmetrische Verschlüsselung) ist C# deutlich abgeschlagen auf dem letzten Platz. Zu beachten ist, dass die Anzahl der Codezeilen nicht nur von der Programmiersprache, sondern auch vom jeweiligen Entwickler beeinflusst ist. Somit ist es durchaus denkbar, dass die Beispiele noch kompakter geschrieben werden können.

4.4 Richtlinien

Die Richtlinien sollen die Qualität der Beispiele erhöhen und dafür sorgen, dass die Anforderungen von CryptoExamples erfüllt werden. Nachfolgend werden die sprachspezifischen Richtlinien für die Erstellung weiterer Beispiele für die Sprache C# behandelt. Die Richtlinien 2, 3, 19, 20 und 21 wurden in Zusammenarbeit mit Lisa-Marie Teis im Rahmen ihrer Bachelorarbeit (“Richtlinien für die Erstellung von CryptoExamples“) [Tei18] erstellt.

4.4.1 Aus den Anforderungen oder von der CryptoExamples Webseite gegeben

Namenskonventionen

1. Die Sprache, in der Bezeichner benannt werden, ist Englisch.
2. Klassennamen beginnen mit dem Wort “Example” gefolgt von dem behandelten Szenario (z. B. ExampleHashing).
3. Die Methode, in der das vollständige Beispiel enthalten ist, wird mit “Demonstrate”, gefolgt vom behandelten Szenario, benannt (z. B. DemonstrateHashing).

Dokumentation

4. Es müssen Zeilenkommentare an den Stellen hinzugefügt werden, an denen der Code ansonsten zu kompliziert oder nicht eindeutig ist.

Code

5. Die Zeichencodierung ist UTF-8.
6. Der Code muss mit der zuletzt veröffentlichten stabilen Version der verwendeten Laufzeitumgebung lauffähig sein.
7. Wird zur Demonstration der Funktionalität ein einzeiliger Text verwendet, dann soll folgender Text abhängig von dem gewählten Szenario gewählt werden.
   - Symmetrische und asymmetrische Verschlüsselung: `"Text that is going to be sent over an insecure channel and must be encrypted at all costs!"`.
   - Hashing: `"Text that should be authenticated by comparing the hash of it!"`.
   - Signing: `"Text that should be signed to prevent unknown tampering with its content."`.
8. Bei einem mehrzeiligen Text soll folgender Text verwendet werden: `"Multiline text:\nMultiline text:\n"`.
9. Der Text soll mit Base64 in Binärdaten kodiert werden und vice versa.
10. Die Ausgaben des Programms werden durch einen Logger behandelt, sodass mögliche Schwachstellen innerhalb von `Console.WriteLine` nicht ausgenutzt werden können oder dass bei einer 1:1 Kopie des Codes, dieser bei dessen Ausführung nichts von seinem Inhalt nach außen gibt (z. B. ein Passwort auf der Konsole).

Die Demonstrate-Methode

11. Für die Szenarien müssen sichere Verfahren und Werte (z. B. für die Schlüssellänge) verwendet werden. Hierfür gibt es mehrere mögliche Quellen, die herangezogen werden können. Zum einen wird auf der GitHub-Seite, auf der die Richtlinien zusammengefasst werden[7], eine fortlaufend aktualisierte Tabelle mit entsprechenden Informationen bereitgestellt. Zum anderen sind dort ebenfalls die Quellen angegeben[8], von denen diese Informationen stammen.
12. Der Code, für das Demonstrieren der Funktionalität, ist innerhalb der Demonstrate-Methode und wird von der Main-Methode aufgerufen.

Fehlerbehandlung

13. Das Abfangen von Fehlern erfolgt am Ende des Codes, um den Beispielcode nicht zu unterbrechen.
14. Fehler werden im Logger protokolliert.
15. Es sollen keine unbestimmten Fehler durch `catch(Exception e)` abgefangen werden.

4.4.2 Neue oder geänderte Richtlinien für C#

Namenskonventionen

16. Die Codekonventionen von Microsoft für C# sollen benutzt werden[9].

Dokumentation

17. Es müssen Klassenkommentare enthalten sein. Diese sollen folgende Informationen beinhalten:
    a) Den Autor des Beispiels.
    b) Einen Link zu weiteren Informationen über die verwendeten Kryptosysteme (z. B. zu Wikipedia).
18. `#region` soll benutzt werden, um logische Elemente des Codes voneinander zu trennen (z. B. `#region Verschlüsselung` und `#region Entschlüsselung`)

[7] CryptoExamples-Guidelines Übersicht von sicheren kryptografischen Algorithmen etc. auf GitHub https://github.com/cryptoexamples/CryptoExamples-Guidelines#overview-of-secure-cryptographic-algorithms-and-parameter-choices (Stand 05.11.2018)

[8] CryptoExamples-Guidelines Quellen für Algorithmen etc. auf GitHub https://github.com/cryptoexamples/CryptoExamples-Guidelines#choosing-secure-algorithms-and-concepts (Stand 05.11.2018)

[9] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions vom 20.07.2015 (Stand 06.11.2018)

Code

19. Der Namespace ist `com.cryptoexamples.csharp.[VERWENDETE_KRYPTO_BIBLIOTHEK]` (z. B. com.cryptoexamples.csharp.bouncycastle).

Die Demonstrate-Methode

20. Die Demonstrate-Methode hat einen sinnvollen Rückgabewert (z. B. bei Hashing der erzeugte Hashwert).
21. Verwendete Werte (z. B. Passwörter) werden der Methode als Parameter übergeben.

Code-Analyse

22. Für die statische Code-Analyse wird SonarLint verwendet[10].

Tests

Die Funktionalität des Codes wird durch Tests sichergestellt (z. B. Unit-Tests).

23. Die Tests befinden sich in einem Testprojekt.
24. Alle Tests sind in einer Klasse deren zugehöriger Dateiname auf "Tests.cs" endet.
25. Um die Funktionalität zu prüfen wird entweder der Rückgabewert der Demonstrate-Methode verwendet oder der Inhalt des Loggers mit einem vorher definierten Sollwert abgeglichen.

4.4.3 Richtlinien vs. Anforderungen

Die Tabelle 4.2 listet die Richtlinien auf und stellt sie den Anforderungen gegenüber. Dabei steht ein ● für positive und ein ○ für negative Auswirkungen auf die Anforderungen (z. B. ein Kommentar, der sich positiv auf die Anforderung der Dokumentation auswirkt, hat gleichzeitig negative Auswirkungen auf die Minimalität, da mehr Codezeilen vorhanden sind). Leere Zellen bedeuten, dass die Richtlinie keinen nennenswerten Einfluss auf die Anforderung hat oder dass dieser nicht bestimmt werden kann.

[10] https://www.sonarlint.org/ (Stand 26.10.2018)

Wie die Bewertung zustande kommt, wird im Folgenden anhand von zwei Beispielen gezeigt. Hierfür werden die Richtlinien 11 und 12 genauer betrachtet.

- Richtlinie 11: Für die Szenarien müssen sichere Verfahren und Werte verwendet werden.
  - sicher: Offensichtlich bewirkt diese Anforderung einen positiven Effekt auf die Sicherheit der Beispiele.
  - minimal: Der Einfluss ist nicht bestimmbar. Dies hängt von der API der Bibliothek ab, die verwendet wird.
  - vollständig: Dadurch, dass die verwendeten Verfahren sicher sind, kann der Code direkt kopiert und verwendet werden, ohne dass eine Anpassung nötig ist.
  - ausführbar: Die Verfahren haben keinen Einfluss auf die Ausführbarkeit.
  - getestet: Sichere Verfahren führen nicht zu einem getesteten Code.
  - dokumentiert: Die Verfahren haben keinen Einfluss.
- Richtlinie 12: Der Code für das Demonstrieren der Funktionalität ist innerhalb der Demonstrate-Methode.
  - sicher: Kein Einfluss.
  - minimal: Der Code wird größer, da eine neue Methode erstellt werden und diese auch noch aufgerufen werden muss.
  - vollständig: Durch die Auslagerung des Bereichs, in dem die komplette Logik stattfindet in eine separate Methode, lässt sich die Logik einfacher kopieren, da kein unnötiger Code enthalten ist.
  - ausführbar: Ob der komplette Code in der Main- oder einer anderen Methode ist, hat auf die Ausführbarkeit keinen Einfluss.
  - getestet: Dass der Code in einer eigenen Methode ist, hat keinen Einfluss.
  - dokumentiert: Kein Einfluss.

Mit Blick auf die Tabelle und einer Wertung jedes ● positiven Effekts mit +1 sowie jedes ○ negativen Effekts mit -1, fällt auf, dass keine Richtlinie in der Summe einen negativen Effekt hat (heißt einen Wert von < 0). Somit kann davon ausgegangen werden, dass alle Richtlinien in Bezug auf die Anforderungen der CryptoExamples-Plattform die Beispiele, wenn nicht verbessern, auf jeden Fall nicht verschlechtern.

Eine Auffälligkeit besteht darin, dass viele Richtlinien die Minimalität verschlechtern. Dafür haben über die Hälfte aller dieser Richtlinien (6/10) mindestens zwei positive Effekte, wodurch davon ausgegangen werden kann, dass mit dem Verletzen der Minimalität im Allgemeinen eine Verbesserung der Beispiele einhergeht.

Ebenfalls auffällig ist die Tatsache, dass die meisten Richtlinien die Vollständigkeit positiv beeinflussen. Dies ist jedoch nicht verwunderlich, da eines der Hauptziele von CryptoExamples ist, Beispiele zu entwickeln, die ohne (große) Anpassungen von Dritten verwendet werden können. Somit ist klar, dass viele Richtlinien die Vollständigkeit verbessern.

Richtlinie	sicher	minimal	vollständig	ausführbar	getestet	dokumentiert
1			●			
2						
3						
4		○	●			●
5			●			
6	●		●	●		
7		○	●			
8		○	●			
9			●			
10	●	○			●	
11	●		●			
12		○	●			
13		○	●		●	
14		○			●	
15		○	●		●	
16			●			
17		○	●			●
18		○	●			●
19			●			
20			●			
21			●			
22	●		●		●	
23					●	
24						
25					●	

Tabelle 4.2: Gegenüberstellung von positiven und negativen Effekten auf die Anforderungen

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Das Ziel dieser Arbeit war es, sichere kryptografische Codebeispiele für CryptoExamples zu entwickeln. Gleichzeitig sollen mit den Erfahrungen und Erkenntnissen, die während der Entwicklung der Beispiele gesammelt wurden, Richtlinien erstellt werden, sodass für die Entwicklung von zukünftigen Beispielen mit C# die Anforderungen von CryptoExamples erfüllt werden.

Dazu wurden in Kapitel 2 wichtige Grundlagen von kryptografischen Verfahren erläutert, sowie die für die Entwicklung der Beispiele verwendeten Technologien.

In Kapitel 3 wurden die Anforderungen *sicher*, *minimal*, *vollständig*, *ausführbar*, *getestet* und *dokumentiert* von CryptoExamples genauer beschrieben. Des Weiteren wurde auf die Szenarien eingegangen, für die Beispiele in der Sprache C# entwickelt wurden. Die beiden letzten Abschnitte beschäftigen sich mit den Nutzergruppen, die mit oder für CryptoExamples arbeiten (siehe Abschnitt 3.2) sowie der Plattform CryptoExamples (Abschnitt 3.3), indem auf die Architektur und Funktionsweise der Plattform eingegangen wird.

Das 4. Kapitel thematisierte unter anderem den Entwurf (siehe Abschnitt 4.1). In dem Abschnitt wurde darauf eingegangen, wie die verwendeten Technologien miteinander arbeiten und welchen Einfluss diese auf die Anforderungen von CryptoExamples haben. In dem nachfolgenden Abschnitt 4.2 wurde auf die Implementierung eingegangen und die Probleme, die es bei dieser gab. Schließlich wurde der Beispielcode für die implementierten Beispiele im Abschnitt 4.3 gezeigt und beschrieben. Außerdem wurde am Ende des Abschnittes auf Auffälligkeiten eingegangen und ein Vergleich der Beispiele von C# mit anderen Programmiersprachen durchgeführt. Schlussendlich wurden die aufgestellten Richtlinien in Abschnitt 4.4 aufgezählt. Diese wurden in allgemeine und sprachspezifische Richtlinien unterteilt. Die allgemeinen Richtlinien waren größtenteils durch die Anforderungen oder von der CryptoExamples Plattform vorgegeben. Die sprachspezifischen beziehen sich auf die Sprache C# wobei nicht ausgeschlossen wird, dass diese teilweise auch für andere Sprachen verwendet werden können. Abschließend wurden in Abschnitt 4.4.3 die Richtlinien mit den Anforderungen in Verbindung gebracht und aufgezeigt wie sie diese beeinflussen.

5.2 Ausblick

Kryptografie wird immer wichtiger und gerät stetig weiter ins Blickfeld der Öffentlichkeit. Deswegen ist eine Plattform wie CryptoExamples von Bedeutung, da sie mit ihren Beispielen dafür sorgt, dass Software sicherer wird. Es ist somit unerlässlich, dass diese ständig weiterentwickelt und aktualisiert werden, was wiederum eine aktive Mitarbeit aller beteiligter und auch außenstehender Personen voraussetzt.

Was der Plattform momentan noch fehlt, sind weitere Beispiele. Vor allem die eingeschränkte Sprachvielfalt ist ein Problem, da sie die potenzielle Nutzerschaft stark einschränkt. Ebenfalls sind bisher keine objektorientierten Beispiele vorhanden, die jedoch durchaus einen Mehrwert bieten würden. Da die Logiken voneinander getrennt wären, würde die Lesbarkeit und Nutzbarkeit verbessert werden. Das wiederum könnte die Integration der Beispiele in die eigene Software deutlich vereinfachen. Neben den Beispielen fehlen außerdem die sprachspezifischen Richtlinien für andere Sprachen. Ein potenzieller Mitwirkender müsste den Mehraufwand der Erstellung dieser Richtlinien übernehmen, was abschreckend wirken könnte.

Aus dieser Arbeit ergeben sich für C# im Speziellen ebenfalls Verbesserungsmöglichkeiten, die zukünftig umgesetzt werden sollten. Zum einen fehlt bisher noch eine Validierung der Beispiele, was zur Folge hat, dass deren Integration auf der CryptoExamples Plattform bisher noch nicht durchgeführt wurde. Zum anderen sollten die erarbeiteten Richtlinien in die CryptoExample-Guidelines aufgenommen werden, damit zukünftige Mitwirkende sich an diesen orientieren können. Des Weiteren könnten Beispiele für weitere kryptografische Bibliotheken erstellt werden, um somit die Auswahl zu vergrößern. Eine Implementierung des Szenarios Schlüsselspeicher steht ebenso noch aus. Außerdem sollte eine Übertragung der Richtlinien auf weitere .Net Sprachen wie VB.Net gut durchführbar sein, wodurch die Sprachvielfalt auf der Webseite weiter vergrößert werden könnte. Schlussendlich sollten zu den vorhandenen Beispielen noch objektorientierte Beispiele erarbeitet werden, sodass die Aufteilung in logische Bestandteile der Szenarien (wie z. B. die Verschlüsselung, Entschlüsselung, ...) durchgeführt werden kann.

Diese Änderungsmöglichkeiten machen deutlich, dass eine regelmäßige Überprüfung, Anpassung und Bewertung der Beispiele und Richtlinien, nicht nur in C#, essenziell ist. Dieser Mehraufwand lässt sich leicht durch die überwiegenden Vorteile, wie beispielsweise die höhere Sicherheit, rechtfertigen.

Sollte es gelingen, weitere Sprachen einzubinden, um die Plattform zu vergrößern und vor allem die Aktualität der Beispiele sicherzustellen, wird CryptoExamples eine der zentralen Anlaufstellen für sichere Codebeispiele werden.

Literaturverzeichnis

[18a] *.NET wird mit .NET Core plattformübergreifend*. Mai 2018. URL: https://msdn.microsoft.com/de-de/magazine/mt694084.aspx (zitiert auf S. 23).

[18b] *Attacken auf deutsche Industrie verursachten 43 Milliarden Euro Schaden*. 13. Sep. 2018. URL: https://www.bitkom.org/Presse/Presseinformation/Attacken-auf-deutsche-Industrie-verursachten-43-Milliarden-Euro-Schaden.html (zitiert auf S. 15, 16).

[18c] *Code examples for common crypto scenarios*. 22. Okt. 2018. URL: https://www.cryptoexamples.com/index.html (zitiert auf S. 29).

[18d] *Java Crypto with JDK - Available Crypto Scenarios and Use Cases*. 22. Okt. 2018. URL: https://www.cryptoexamples.com/java_landing_page.html (zitiert auf S. 30).

[18e] *Java Password Based String Encryption using JDK*. 22. Okt. 2018. URL: https://www.cryptoexamples.com/java_string_encryption_password_based_symmetric.html (zitiert auf S. 31).

[18f] „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“. In: Bundesamt für Sicherheit in der Informationstechnik, 29. Mai 2018 (zitiert auf S. 19–21).

[18g] *Main-Repository of CryptoExamples section Architecture*. 6. Feb. 2018. URL: https://github.com/cryptoexamples/CryptoExamples%5C#architecture (zitiert auf S. 28).

[18h] *Microsoft to acquire GitHub for $7.5 billion*. 4. Juni 2018. URL: https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/ (zitiert auf S. 24).

[18i] *The State of the Octoverse*. 2018. URL: https://octoverse.github.com/ (zitiert auf S. 24).

[GIJ+12] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, V. Shmatikov. „The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software“. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: ACM, 2012, S. 38–49. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382204. URL: http://doi.acm.org/10.1145/2382196.2382204 (zitiert auf S. 16).

[GS16] M. Green, M. Smith. „Developers are Not the Enemy!: The Need for Usable Security APIs“. In: *IEEE Security Privacy* 14.5 (Sep. 2016), S. 40–46. ISSN: 1540-7993. DOI: 10.1109/MSP.2016.111 (zitiert auf S. 16).

[Hir18] T. Hirzel. „Erstellung von CryptoExamples in JavaScript“. In: 2018 (zitiert auf S. 16).

[Klo18] M. Kloppenburg. „Erstellung von CryptoExamples in Python“. In: 2018 (zitiert auf S. 16).

[KW11] R. Küsters, T. Wilke. „Moderne Kryptographie“. In: *Vieweg+ Teubner* 4 (2011) (zitiert auf S. 19).

[MW18] K. Mindermann, S. Wagner. „Usability and Security Effects of Code Examples on Crypto APIs - CryptoExamples: A platform for free, minimal, complete and secure crypto examples“. In: *Proceedings of the 16th Annual Conference on Privacy, Security and Trust (PST)*. Belfast, Northern Ireland, United Kingdom: IEEE, 28. Aug. 2018 (zitiert auf S. 16, 27).

[Tei18] L.-M. Teis. „Richtlinien für die Erstellung von CryptoExamples“. In: 2018 (zitiert auf S. 16, 44).

Alle URLs wurden zuletzt am 20. 11. 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