Bachelorarbeit
zur Erlangung des Grades Bachelor of Science Informatik

Entwicklung einer Web-Anwendung zur Veranschaulichung des Verifizierungsprozesses von Zertifikaten unter Berücksichtigung verschiedener Gültigkeitsmodelle

Betreuer	Jendrik Winkel und Prof. Bernhard Esslinger
Erstprüfer	Prof. Bernhard Esslinger
Zweitprüfer	Prof. Dr. Roland Wismüller
vorgelegt von	Thorben Meiswinkel
Matrikelnummer	1501616
Abgabedatum	27. Juli 2023 (Update 29.09.2023)

Kurzzusammenfassung

Das Ziel dieser Bachelorarbeit ist, die Entwicklung einer Web-Applikation, um die Verifikation von Zertifikaten unter Berücksichtigung verschiedener Gültigkeitsmodelle didaktisch zu veranschaulichen. Dafür soll die Applikation so gebaut sein, dass sie einen übersichtlichen und interaktiven Blick auf die Gültigkeitsprüfung der Signatur gibt, damit verständlich wird, warum es vorkommen kann, dass eine Signatur – ohne weitere Einwirkungen – an einem Tag als gültig und am nächsten Tag als ungültig gewertet wird. Dazu soll es die Möglichkeit geben, die im Verifikationsprozess benutzten Parameter selbst einzustellen oder zu laden: diese sind die Zertifikate, deren Gültigkeitszeiträume, zu signierende Dateien und Nutzerschlüssel.

Abstract

The aim of this bachelor thesis is to develop a web application to didactically illustrate the verification of certificates taking into account different validity models. For this purpose, the application should be built in such a way that it gives a clear and interactive view of the validity check of the signature, so that it becomes understandable why it can happen that a signature – without further intervention – is evaluated as valid on one day and as invalid on the next day. For this purpose, it should be possible to set or load the parameters used in the verification process: these are the certificates, their validity periods, files to be signed and user key.

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel stellt die Einleitung in die Bachelorarbeit dar. Es werden die Motivation und die Ziele erläutert. Außerdem werden der Aufbau der Arbeit und die benutzten Tools näher beschrieben.

1.1 Motivation

Jeden Tag kommen mehr Maschinen in allen Bereichen des modernen Lebens zum Einsatz. Dadurch steigt die Wichtigkeit für elektronische Kommunikation, sowohl zwischen Menschen als auch Maschinen. Diese Tatsache macht aber die Kommunikation zu einem attraktiven Ziel für Angreifer. Um Angriffe zu erkennen, muss es eine Möglichkeit geben der elektronischen Kommunikation zu vertrauen. Für das Vertrauen in die elektronische Kommunikation kommt Kryptografie zum Einsatz. In ihren Protokollen und den Public-Key-Infrastrukturen werden Zertifikate und Signaturen verwendet. Signaturen sollen die Integrität der Daten sichern. Zertifikate sollen sicherstellen, dass der öffentliche Schlüssel zur Verifikation der Signatur von der Entität stammt, die die Signatur erstellt hat.

Unter Umständen kann es aber vorkommen, dass die Überprüfung einer Signatur an zwei unterschiedlichen Zeitpunkten einmal als gültig und einmal als ungültig erkannt wird, was verschiedenste Gründe haben kann. Gewollt ist, dass Änderungen an der Signatur oder der signierten Datei erkannt werden – unabhängig davon, ob die Änderung durch einen technischen Fehler oder durch einen Angriff erfolgte. Neben technischen Fehlern und Angriffen kann auch ein ungültiges Zertifikat in der Zertifikatskette eine mögliche Ursache für eine ungültige Signatur sein. Ein Zertifikat kann ebenfalls aus mehreren Gründen ungültig sein: entweder ist seine Gültigkeit abgelaufen, es wurde widerrufen oder es wurde verändert. Das Ergebnis der Gültigkeitsprüfung von Zertifikatsketten kann nach unterschiedlichen Modellen verschieden ausfallen. Sowohl eine fehlerhafte Signatur als auch ein Problem in der Zertifikatskette führen zu einer Warnung über eine falsche Signatur. Um mit einer solchen Warnung korrekt umzugehen, ist ein Grundverständnis für das Thema notwendig.

Diese Arbeit beschäftigt sich mit der Erstellung einer Web-Applikation, um dieses Problem didaktisch aufzuarbeiten und für jeden Nutzer verständlich zugänglich zu machen. Hierzu wurde eine Web-App für CrypTool-Online (CTO) entwickelt: Die Web-App ermöglicht es, mit Zertifikaten, deren Gültigkeitszeiträumen und mit digitalen Signaturen zu experimentieren.

1.2 Ziele

Das Ziel dieser Bachelorarbeit war die Erstellung einer Web-App, die dazu dienen soll, die Prinzipien der Gültigkeitsprüfung von Signaturen und Zertifikatsketten anschaulich darzustellen. Hierzu können verschiedene Zertifikate in die Web-App im CTO geladen werden. Über Schieberegler und Date-Picker können dafür unterschiedliche Gültigkeitszeiträume eingestellt werden. Besonders die Gültigkeiten nach unterschiedlichen Modellen sollen so veranschaulicht werden. Einzelne Komponenten sind in dem User Interface (UI) farblich hervorgehoben, sodass genau erkannt werden kann, an welchem Punkt ein Fehler auftritt. Zusätzlich ist es möglich, eine eigene Datei zu laden und deren Signatur zu berechnen, um damit anschließend eine Dateiänderung simulieren zu können.

1.3 Aufbau der Arbeit

Die Bachelorarbeit besteht aus fünf verschiedenen Kapiteln. Dabei stellt Kapitel 1 die Einleitung in die Arbeit dar.

In Kapitel 2 werden die nötigen Grundlagen für das Verständnis des Projekts erläutert. Es werden die benutzten Bibliotheken und das Dachprojekt CTO eingeführt sowie die Grundlagen für digitale und elektronische Signaturen und deren Gültigkeit erklärt.

Anschließend wird in Kapitel 3 der Prozess der Entwicklung beleuchtet. Zunächst wird analysiert, welche Anforderungen an die Web-App gestellt werden. Darauf folgend wird die Entwicklung des UI dargestellt. Danach wird das jetzige UI beschrieben und dessen Funktion genauer erläutert. Abschließend in diesem Kapitel werden die Funktionen der Web-App erklärt.

Kapitel 4 enthält die Evaluation der geleisteten Arbeit. Es wird auf Probleme während der Entwicklung eingegangen und die benutzten Bibliotheken werden bewertet.

Am Abschluss der Arbeit stehen in Kapitel 5 eine Zusammenfassung und der Ausblick auf zukünftige Erweiterungen und Verbesserungen der Web-App.

1.4 Benutzte Tools

Der Code für diese Arbeit wurde mit verschiedenen Tools auf der Plattform Windows 10 entwickelt. Das Schreiben des Codes fand in Visual Studio Code statt. Visual Studio Code enthält die Extension „Prettier“, welche den Code des gesamten CTO einheitlich formatiert halten soll. Das React Framework ist für die Entwicklung zum Einsatz gekommen. Die grafische Oberfläche wurde mit Chakra-UI-Komponenten designt und mit Icons der FontAwesome-Kollektion versehen. Übersetzungen in der Web-App sind von DeepL[1] übersetzt worden. Die verwendeten Krypto-Funktionen wurden mit Hilfe der node-forge-Bibliothek implementiert. Die bereitgestellten Zertifikate und der private Schlüssel generierte OpenSSL in der Desktopanwendung. Für die gesamte Verwaltung des Projekts ist GitHub zum Einsatz gekommen. Clientseitig ist Github Desktop benutzt worden. Dieses Tool ist von GitHub als UI-Version für Windows und MacOS entwickelt worden, um die Benutzung zu vereinfachen.

[1] https://www.deepl.com/translator

2 Grundlagen

Dieses Kapitel bildet die Grundlage für diese Arbeit. Es wird eine Einführung in das CrypTool (CT)-Projekt gegeben. Außerdem werden digitale und elektronische Signaturen sowie deren Unterschiede und Gemeinsamkeiten beleuchtet. Wie das Gültigkeitsmodell die Prüfung der Signaturen beeinflusst wird erklärt und eine kurze Einführung in die benutzten Frameworks wird gegeben.

2.1 Das CrypTool-Projekt

CT ist ein Open-Source-Projekt, das Kryptografie und Kryptoanalyse als Lernsoftware für Schulen, Hochschulen, Unternehmen und in der Ausbildung zur Verfügung stellt. Ziel des CT ist es, das Verständnis für Kryptologie zu erhöhen und die Benutzer entsprechend zu schulen. Das CT-Projekt begann 1998 mit der Entwicklung des CrypTool 1 (CT1), welches in C++ und MFC geschrieben wurde und nur für Windows verfügbar ist. Die Weiterentwicklung von CT1 ist mittlerweile eingestellt. Das heißt, es kommen keine neuen Funktionsupgrades hinzu, die Wartung allerdings ist noch nicht eingestellt.

Das Folgeprojekt von CT1, CrypTool 2 (CT2), wurde in den Programmiersprachen C# und .NET entwickelt. Im Gegensatz zum CT1 ist im CT2 eine visuelle Programmiersprache umgesetzt worden und es setzt einen Fokus auf die Analyse klassischer Verfahren. Da die beiden Versionen ausschließlich unter Windows benutzbar sind, wurde mit JavaCrypTool (JCT) eine Software für alle großen Betriebssystem-Plattformen (Windows, Linux, macOS) entwickelt. JCT verfügt über eine dokumenten- und eine funktionsorientierte Sicht und legt einen Schwerpunkt auf digitale Signaturen.

Neben diesen drei Software-Programmen gehören zu CT noch zwei Online-Projekte: Das erste dieser Projekte, MysteryTwister (MTC3), ist ein Wettbewerb in einem Quiz-Format mit Aufgaben (Challenges) aus der Kryptologie. Auf Basis der Schwierigkeit der Aufgaben werden Punkte vergeben, welche in einer Bestenliste und der Nutzerstatistik zu finden sind. Das andere Projekt ist CrypTool-Online (CTO). Mit seinen direkt im Browser ablaufenden kryptografischen Verfahren ermöglicht das CTO eine Verwendung ohne die Notwendigkeit eines Downloads. Ein weiteres Ziel von CTO ist es, Kryptografie besonders für jüngere Menschen erreichbar und interessant zu gestalten. [1]

2.2 Digitale Signaturen

Eine digitale Signatur soll den Ursprung einer Nachricht nachvollziehbar machen. Dabei soll sowohl der Empfänger der Nachricht nachvollziehen können, wer die Nachricht

geschickt hat, als auch der Sender soll vor Dritten seinen Urhebernachweis erbringen können. Außerdem kann so auch die Unversehrtheit der Nachricht sichergestellt werden. Eine Signatur wird erzeugt, indem die Nachricht M mit einem Hash-Algorithmus (z.B. SHA-256) gehasht wird und danach der Hashwert H1 mit dem privaten Schlüssel durch einen Signaturalgorithmus (z.B. RSA-FDH) zur Signatur S1 berechnet wird. Der Signaturalgorithmus ist ein asymmetrischer Algorithmus, der den privaten Schlüssel des Senders nutzt.

Der Empfänger der Nachricht M überprüft nun die Signatur S1, indem er die Nachricht M selbst hasht und daraus den Hashwert H2 erhält. Außerdem wendet er den Signaturalgorithmus auf die Signatur S1 an, aber verwendet den öffentlichen Schlüssel des Senders. Das sollte den Hashwert H1 liefern. Wenn H1 mit H2 übereinstimmt, ist die Signatur der Datei korrekt. Weil Signaturverfahren rechtlich verbindlich sind (und in der Vergangenheit Signaturalgorithmen schon gebrochen wurden), gibt es gesetzliche Vorgaben für die zu verwendenden Hashalgorithmen und Schlüssellängen. [2][3]

Welche Signaturalgorithmen als sicher anerkannt werden hängt von dem Land ab, in dem sich die Kommunikationspartner befinden. Das National Institute of Standards and Technology (NIST)[2] hat „DSA“, „RSA“ und „ECDSA“ (Stand 2.6.2023) zur sicheren Verwendung genehmigt. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) nennt in der Technischen Richtlinie über Empfehlungen und Schlüssellängen[3] „RSA“, „DSA“, „DSA“-Varianten auf elliptischen Kurven und „Merkle“-Signaturen mit den vorgegebenen Hash-Funktionen und nötigen Schlüssellängen (Stand 9.1.2023).

2.2.1 Zertifikate und Public-Key-Infrastruktur

Zertifikate werden verwendet um sicherzustellen, dass ein öffentlicher Schlüssel zu einer bestimmten Entität gehört (z.B. einer Person oder einem Unternehmen oder einer Maschine in der Produktion). Ein Zertifikat besteht aber nicht nur aus einem öffentlichen Schlüssel, sondern aus unterschiedlichen Bestandteilen. Nach dem X.509-Standard [4] besteht ein Zertifikat aus der Seriennummer, dem Subjekt, für wen es ausgestellt wurde, der ausstellenden Certificate Authority (CA), der Information, ob es sich selbst signiert hat, der Version des Zertifikats und dem Gültigkeitszeitraum sowie einer digitalen Signatur, die dem gesamtem Zertifikat Integrität verleiht. Zusätzlich können Erweiterungen hinzugefügt werden, wie ein alternativer Name des Subjekts, eine Einschränkung der Schlüsselnutzung, wo die Certificate-Revocation-List (CRL) zu finden ist, und einige mehr [5].

[2] https://csrc.nist.gov/Projects/digital-signatures
[3] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=8

Allerdings muss auch die Integrität des Zertifikats gesichert werden, weil auch das Zertifikat abgefangen oder verändert werden kann. Entweder geschieht dies über einen sicheren Weg (wie z.B. ein Telefonat, in dem die Fingerabdrücke der Zertifikate überprüft werden [6]) oder es wird eine Public-Key-Infrastruktur (PKI) verwendet. Eine PKI benötigt eine vertrauenswürdige dritte Instanz, die für die Kommunikationsteilnehmer als Wurzel-Zertifikat agiert. Auf Basis dieses Wurzel-Zertifikats, welches als Trust Anchor fungiert, kann nun die gesamte Infrastruktur aufgebaut werden. Mit dem privaten Schlüssel des Wurzel-Zertifikats können weitere Zertifikate signiert und damit legitimiert werden. Die neuen Zertifikate gehören entweder direkt einem Nutzer für die Kommunikation oder aber einer weiteren Zertifizierungsstelle, einer sogenannten CA. Eine CA kann wieder weitere Zertifikate ausstellen, die für einen Nutzer oder untergeordnete CAs verwendbar sind. Eine CA sollte Zertifikate nur ausstellen, nachdem sie die Identität des Antragstellers überprüft hat (der Antragsteller erstellt einen Certificate Request und weist sich bspw. durch Vorlage des Personalausweises bei der zuständigen Registrierungsstelle aus). Ohne diese ordentliche Identitätsüberprüfung verliert die ganze Infrastruktur ihre Vertrauenswürdigkeit.

Für die Kommunikation zwischen zwei Teilnehmern werden nun die Zertifikate ausgetauscht und können von jedem überprüft werden. Dafür wird in den Zertifikaten nach dem Aussteller geschaut und mit dem öffentlichen Schlüssel des Ausstellers die Signatur des Zertifikats überprüft. Dieser Vorgang wiederholt sich, bis entweder ein Zertifikat oder eine Signatur als ungültig befunden oder der Trust Anchor erreicht wird. Wird der Trust Anchor erreicht, wird das Zertifikat des Nutzers akzeptiert und es kann mit der Verschlüsselung begonnen werden. Falls nicht, wird das Zertifikat nicht akzeptiert und das Problem muss identifiziert und behoben werden.

Für die Identifizierung des Problems muss bekannt sein, was alles zur Verwerfung führen kann: Erstens kann das Zertifikat verändert worden sein, dabei ist es egal, ob dies durch einen Angreifer oder einen technischen Fehler geschah. Zweitens kann das Zertifikat noch nicht bzw. nicht mehr gültig sein, in beiden Fällen wäre das Zertifikat ungültig. Drittens ist es möglich, dass das Zertifikat auf der CRL seiner CA steht und damit widerrufen wurde. Es ist aber genauso möglich, dass das Zertifikat einer CA auf einer CRL einer höheren Instanz steht, was auch zur Ablehnung des Nutzer-Zertifikats führt.

Die Behebung des Problems ist nicht immer möglich. Eine Veränderung des Zertifikats kann durch erneute Sendung bereits gelöst sein. Falls nicht, sollte das Zertifikat über einen anderen Kanal zu den Kommunikationspartnern gelangen. Bei einem falschen Gültigkeitszeitraum kann nur ein neues Zertifikat Abhilfe schaffen. Außer man würde warten, bis das Zertifikat gültig wird – im Falle, dass es noch nicht gültig war. Wenn ein Zertifikat der Kette auf einer CRL steht, hilft nur ein neues Zertifikat oder am besten eine komplett neue Kette an Zertifikaten. [4][6]

2.2.2 Gültigkeitsmodelle

Neben der Korrektheit der Signatur von Datei und Zertifikaten muss auch das Gültigkeitsmodell berücksichtigt sein. Im Folgenden werden drei der meist verwendeten Modelle vorgestellt.

Schalenmodell Das Schalenmodell kommt häufig in PKI-Applikationen vor und ist im RFC 5280 [7] spezifiziert. Nach dem Schalenmodell wird die Signatur als gültig erkannt, wenn das Datum der Verifikation innerhalb der Gültigkeitszeiträume aller Zertifikate der Kette liegt (siehe Abbildung 1). Der Standard verlangt dabei nicht, dass das Zertifikat des Nutzers zum Signierzeitpunkt gültig ist. Vermutlich liegt das daran, dass das Modell für den Handshake im Internet Protokoll zum Einsatz kommt und damit der Signaturerstellungszeitpunkt nicht gespeichert oder ausgewertet wird. [8, Seite 258ff.]

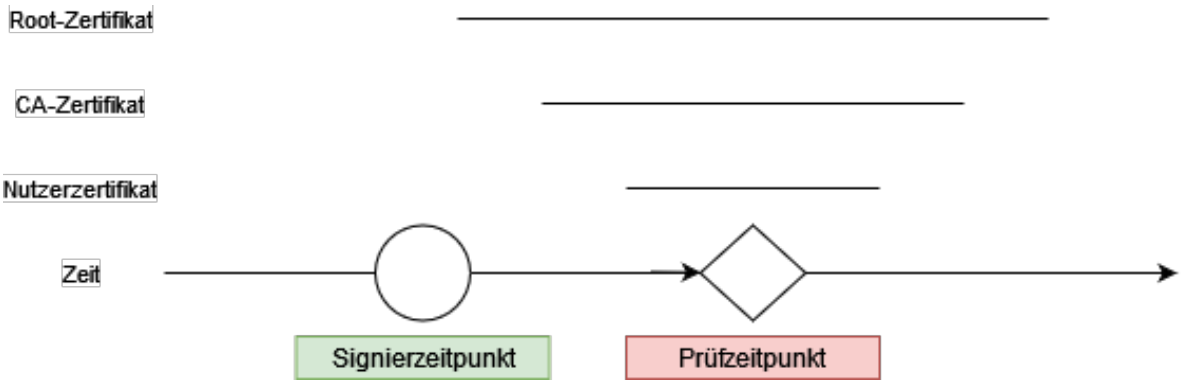

Abbildung 1: Beispiel für das Schalenmodell [9]

Ein Problem dabei ist, dass eine gültige Signatur über die Zeit ihre Gültigkeit verliert. Das ist besonders für Verträge o.Ä. ein Risiko, weil dadurch Verträgen über ihre Laufzeit die Rechtskraft abhanden kommt, wenn der Prüfzeitpunkt z.B. nach dem Ende der Gültigkeit des Teilnehmer-Zertifikats liegt.

Kettenmodell Das Kettenmodell ist im deutschen Signaturgesetz aufgekommen und sollte das Problem mit dem Auslaufen der Gültigkeit einer validen Signatur beheben.[10] Dafür muss immer der Zeitpunkt der Signaturerstellung (nicht der der Signaturprüfung) im Gültigkeitszeitraum des höheren Zertifikats liegen. Also muss zum Zeitpunkt des Signierens der Datei das Nutzerzertifikat gültig sein, dann muss das Datum der Signatur des Nutzerzertifikats innerhalb des Gültigkeitszeitraums des CA-Zertifikats liegen, die Signatur des CA-Zertifikats muss dann im Gültigkeitszeitraum des Wurzel-Zertifikats liegen (siehe Abbildung 2).[11, Zertifikat-Verifikation]

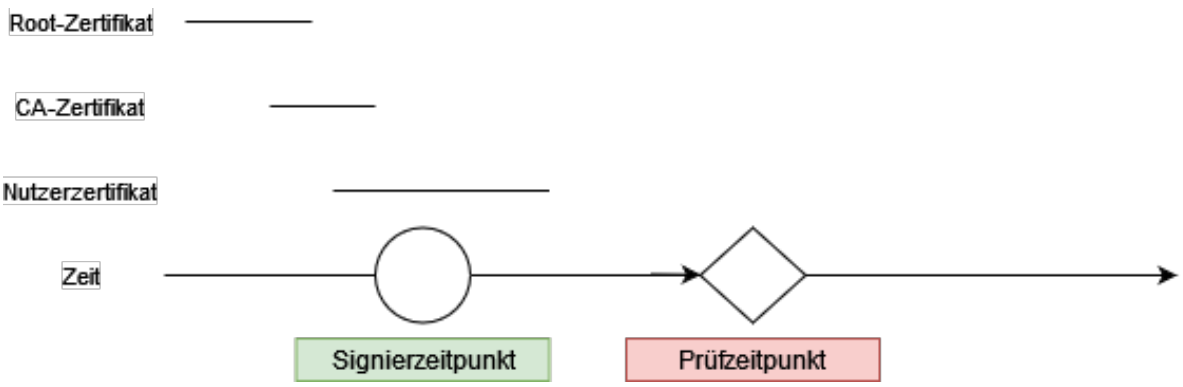

Abbildung 2: Beispiel für das Kettenmodell [9]

Allerdings wirft das auch wieder ein Problem auf. So kann nach diesem Modell jemand, der im Besitz des privaten Schlüssels ist, Zertifikate oder Signaturen zurückdatieren und es würden die Zertifikate und Signaturen als gültig erkannt – trotz ihrer eigentlichen Ungültigkeit.[10]

Erweitertes Schalenmodell Das erweiterte Schalenmodell ist, wie der Name schon sagt, eine Erweiterung des Schalenmodells, weil sich in diesem Modell nicht das Datum der Verifikation sondern der Signaturerstellung in den Gültigkeitszeiträumen aller Zertifikate der Kette befinden muss (siehe Abbildung 3). Es wird manchmal auch als Hybridmodell bezeichnet, weil es quasi die Mischung aus dem Schalenmodell der RFC 5280 und dem Kettenmodell des Signaturgesetzes ist.[8]

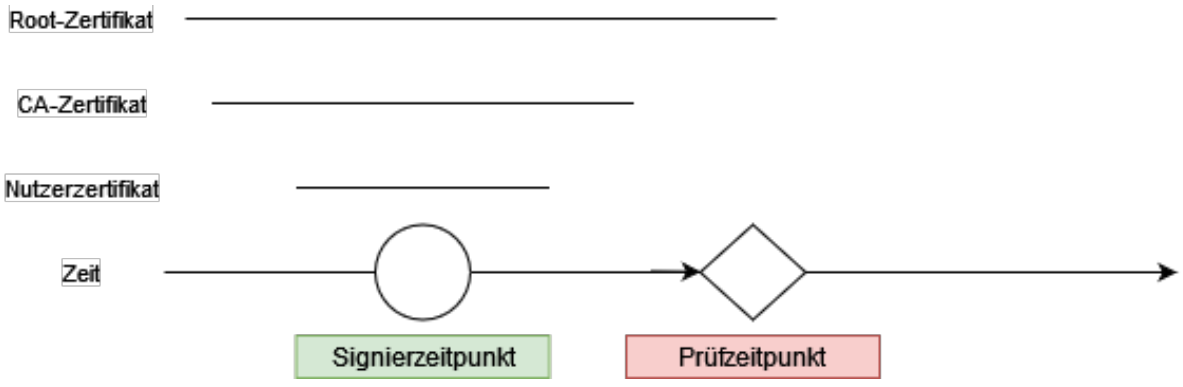

Abbildung 3: Beispiel für das erweiterte Schalenmodell [9]

Das bringt den Vorteil, dass damit dauerhaft gültige Signaturen erstellt werden können. Sogar ohne die Problematik des Zurückdatierens zu besitzen. Aber ein Nachteil des Modells ist es, dass die Gültigkeit der Signatur nun von den Zeiträumen aller Zertifikate der Kette abhängig ist.[12]

2.2.3 Elektronische Signaturen

Eine elektronische Signatur ist der elektronische Ersatz für eine von Hand geschriebene Unterschrift. Die dafür notwendigen Rechtsgrundlagen stehen in der EU-Verordnung eIDAS-Nr. 910/2014. [13] Elektronische Signaturen sind immer nur natürlichen Personen zugeordnet. Für Unternehmen und Behörden gibt es Elektronische Siegel. Beides wird hauptsächlich in der Justiz und öffentlichen Verwaltung verwendet, wobei die Nutzung in der Privatwirtschaft zunimmt. [14]

Der Begriff elektronische Signatur ist nicht gleichzusetzen mit einer digitalen Signatur. In der Kryptografie bezeichnet der Begriff „digitale Signatur“ eine ganze Reihe von Verfahren. In der Softwaretechnik werden digitale Signaturen als Sammelbegriff für Identifikationen verwendet. Elektronische Signatur ist ein rechtlicher Terminus, der zuerst in der EU-Verordnung Art. 3 Nr. 10 eIDAS verwendet wurde. Damit werden mehr als die kryptografischen Verfahren eingeschlossen, z.B. eine eingescannte händische Unterschrift. [14][15]

Es existieren drei verschiedene Arten von elektronischen Signaturen, die Simple Electronic Signature (SES), die Advanced Electronic Signature (AES) und die Qualified Electronic Signature (QES), welche jeweils unterschiedliche Sicherheitsstufen gewährleisten:

Elektronische Signatur Als „Elektronische Signatur“, oder auch SES, bezeichnet man „Daten in elektronischer Form, die anderen elektronischen Daten beigefügt oder logisch mit ihnen verbunden werden und die der Unterzeichner zum Unterzeichnen verwendet“. [16, Art.3 Nr.10] Das kann eine eingescannte Unterschrift oder die bloße Nennung des Namens sein. [13]

Fortgeschrittene elektronische Signatur Eine fortgeschrittene elektronische Signatur, oder auch Advanced Electronic Signature (AES), ist eine Simple Electronic Signature (SES), die die folgenden Punkte erfüllt: [16, Art.26]

- Sie ist eindeutig dem Unterzeichner zugeordnet.
- Sie ermöglicht die Identifizierung des Unterzeichners.
- Sie wird unter Verwendung elektronischer Signaturerstellungsdaten erstellt, die der Unterzeichner mit einem hohen Maß an Vertrauen unter seiner alleinigen Kontrolle verwenden kann.

- Sie ist so mit den auf diese Weise unterzeichneten Daten verbunden, dass eine nachträgliche Veränderung der Daten erkannt werden kann.

Diese Anforderungen sind z.B. mit einer PGP-Signatur erfüllt, weil der auf der Festplatte gespeicherte Signaturschlüssel als Prüfschlüssel der Signatur fungieren kann und dem Inhaber der Festplatte zugeordnet werden kann. Vor Gericht kann diese Signatur nur als Beweis verwendet werden, wenn nachgewiesen wird, dass die Signatur und das Erkennungsmerkmal authentisch sind. [15]

Qualifizierte elektronische Signatur Als „qualifizierte elektronische Signatur“, bzw. QES bezeichnet man „eine fortgeschrittene elektronische Signatur, die von einer qualifizierten elektronischen Signaturerstellungseinheit erstellt wurde und auf einem qualifizierten Zertifikat für elektronische Signaturen beruht“. [16, Art.3 Nr.12] Die qualifizierte Signaturerstellungseinheit ist eine Soft- oder Hardware zur Erstellung einer elektronischen Signatur, welche den Anforderungen des Anhang 2 der EU-Verordnung entspricht. Ein qualifiziertes Zertifikat ist ein von einer qualifizierten Zertifizierungsstelle ausgestelltes Zertifikat und muss den Anforderungen aus Anhang 1 der EU-Verordnung genügen. Eine qualifizierte Zertifizierungsstelle ist eine Zertifizierungsstelle, die von der Aufsichtsstelle (z.B. die Bundesnetzagentur [15]) verifiziert und zertifiziert wurde. [16]

Die qualifizierte Signatur hat sich – zumindest bei Privatpersonen – nicht durchgesetzt. Die einmaligen Kosten für die Anschaffung der erforderlichen Hard- und Software (Kartenleser und Signatursoftware) schreckten ab und es gab keine Vorteile für den Signaturersteller. Evtl. wird sich das ändern mit der durch EU-Gesetzgebung ermöglichten „Fernsignatur“: Damit kann eine qualifizierte elektronische Signatur auch ohne Signaturkarte erstellt werden. Der Vorteil des Verfahrens liegt darin, dass keine zusätzliche technische Ausstattung (Signaturkarte, Lesegerät) für das Erstellen einer qualifizierten elektronischen Signatur benötigt wird. Die unterzeichnende Person muss dafür gegenüber dem Vertrauensdiensteanbieter ihre Identität sicher nachweisen. Der Vertrauensdiensteanbieter erstellt dann die Signatur im Auftrag der unterzeichnenden Person. Ein Beispiel dafür ist der staatliche Online-Ausweis.[4]

2.3 React.js Framework

React ist eine JavaScript (JS)-Bibliothek, die speziell zur Entwicklung von Webseiten geschrieben wurde. In React wird das gesamte UI in einzelne Teile aufgespalten, die Components. Die Components besitzen ein Markup, welches mit der Syntaxerweiterung JSX geschrieben wird. JSX ähnelt sehr Hypertext Markup Language (HTML), ist aber

[4] https://www.personalausweisportal.de/Webs/PA/DE/wirtschaft/eIDAS-konforme-fernsignatur/eidas-konforme-fernsignatur-node.html

strikter als HTML und kann dynamische Informationen enthalten. Für bestehenden HTML-Code gibt es einen Konverter das „<></>“-Tag, welches im Normalfall ein „ “-Element rendert. In JSX ist es auch möglich JS-Code zu schreiben, indem der Code in „{}“ eingeklammert wird. Das bringt den Vorteil, dass das Markup und die Logik in der gleichen Datei geschrieben sind und Werte dynamisch codiert und geändert werden können. Components können auch Eigenschaften (props) übergeben werden, um z.B. bestimmte Teile nur unter Bedingungen anzuzeigen oder Variablen an andere Components zu übergeben, die diese Variablen dann verwenden. [17]

2.4 node-forge-Bibliothek

Node-forge ist eine Bibliothek, die kryptografische Algorithmen in nativem JS implementiert. Diese Bibliothek wurde ausgewählt, weil sie eine der performantesten ist, dennoch einfach zu nutzen und die für diese Arbeit wichtigen Elemente enthält. Besonders hervorzuheben sind die folgenden Kapitel. Das Kapitel X.509 [18, S. X.509] wird verwendet, um die Zertifikate in JS zu repräsentieren und ihnen die Funktion eines Zertifikats zu geben. Außerdem ist in dem Kapitel eine Funktion zu finden mit welcher eine Zertifikatskette verifiziert werden kann. Allerdings waren die Fehlermeldungen bei einer ungültigen Zertifikatskette nicht genau genug, es wurde nicht angezeigt, bei welchem Zertifikat welches Problem vorlag. Aus diesem Grund wurde eine eigene Überprüfungsmethode implementiert. Ein weiteres benötigtes Kapitel war das SHA-256 [18, SHA-256], was das Hashen von Dateien mit dem „SHA-256“-Algorithmus ermöglicht. Der daraus resultierende Hashwert kann anschließend mit einem privaten Schlüssel signiert werden. Die Repräsentation des Schlüssels erfolgt in dieser Arbeit durch das RSA-Kapitel [18, RSA] und dessen Methoden und Objekte. [18]

2.5 React-Komponenten mit Chakra-UI

Die Chakra-UI Bibliothek ist eine Bibliothek für das Entwickeln von React-Apps. Chakra-UI ist ein hauptsächlich Community getragenes Projekt mit einem kleinen festen Team, welches über Sponsoren und Spenden finanziert wird. Die Bibliothek stellt React-Komponenten zum Import zu Verfügung, die schon fertig gebaut sind, sodass nur die eigene Logik hinzugefügt werden muss. Das bringt den Vorteil, dass die Entwicklung des UI weniger Zeit in Anspruch nimmt, bei einem Open-Source-Projekt das Aussehen einheitlich ist und für Entwickler sorgt, dass HTML und Cascading Style Sheets (CSS) Kenntnisse nicht in einem hohen Maße nötig sind. [19]

3 Entwicklung der Web-Applikation

In diesem Kapitel wird die Entwicklung der Web-App zur Veranschaulichung des Verifikationsprozesses von Zertifikaten genauer erläutert. Dazu wird erst analysiert, was die Web-App zu leisten hat und wie sie gestaltet sein soll, damit den Nutzern eine gute Nutzbarkeit bereitgestellt wird. Im Anschluss wird die Entwicklung des UI beschrieben und das Endresultat vorgestellt. Darauf folgend werden die Funktionen und deren Umsetzung erklärt.

3.1 Analyse der Web-Applikation

Die Web-App soll den Verifikationsprozess einer Signatur unter Berücksichtigung der zugrundeliegenden Zertifikate und deren Gültigkeitszeiträume nach unterschiedlichen Modellen veranschaulichen. Dazu werden diverse Funktionen benötigt. Außerdem soll das UI entsprechend gestaltet sein.

Benötigte Funktionen Damit der Prozess der Verifikation erklärt werden kann, muss zuerst die Verifikation als solche implementiert werden. Weil in der Web-App nicht einfach nur Zeitpunkte verglichen werden sollen, muss eine „echte“ Prüfung simuliert werden. Für diese Prüfung werden Zertifikate benötigt. Im Falle der Web-App werden drei Zertifikate verwendet, die in der Web-App benutzbar sein müssen. Eine weitere Vorgabe ist die Veränderbarkeit der Gültigkeitszeiträume der Zertifikate, um unterschiedliche Szenarien ausprobieren zu können, ohne für jeden beliebigen Zeitpunkt ein neues Zertifikat erstellen zu müssen. Außerdem muss eine Signatur erstellt und nach drei verschiedenen Gültigkeitsmodellen verifiziert werden können. Eine weitere Funktion soll die Überprüfung der Zertifikatskette sein, um ein falsches Zertifikat zu identifizieren und den Fehler auszugeben.

Diese Grundfunktionen müssen aber noch erweitert werden, da es möglich sein soll, Dateien in die Web-App zu laden. Die Dateien sollen anschließend verwendet werden können, um den gesamten Prozess zu veranschaulichen.

Dafür muss die Möglichkeit gegeben sein, drei Zertifikate zu laden und diese für den Code nutzbar zu machen. Außerdem soll es für das Nutzerzertifikat noch die Möglichkeit geben, den privaten Schlüssel in Form einer PEM-Datei zu laden, weil dieser für die Signaturerstellung benötigt wird. Zu guter Letzt soll eine beliebige Datei geladen werden können, die für die Signaturerstellung und vor allem für die Signaturprüfung herangezogen werden soll.

Anforderungen an die Oberfläche Neben den Funktionen muss das UI ansprechend designt sein. Die Oberfläche soll intuitiv gestaltet sein, damit jeder Nutzer das UI verstehen und benutzen kann. Dazu werden im UI die Teile, die zur Einstellung der Parameter verwendet werden, räumlich von den Teilen, die für die Überprüfung der Signatur und der Zertifikatskette zuständig sind, getrennt. Es soll auch ersichtlich sein, welcher Parameter eingestellt wird, damit keine Missverständnisse beim Nutzer auftreten. Außerdem soll sich das UI auch in das Gesamtbild des CTO einfügen, um ein harmonisches Gesamtbild zu erzeugen. Weil die App im Web direkt verfügbar ist, soll sich das Design an die Größe des benutzten Bildschirms anpassen.

Auflistung der Anforderungen in Tabellenform

Kategorie	Beschreibung	Details
UI	Nutzerfreundlichkeit	sind die Komponenten sinnvoll gruppiert
UI	Interaktivität	wie viel kann der Nutzer selbst einstellen
UI	Passend zum CTO	sieht die Oberfläche den bestehenden ähnlich
UI	Zweisprachigkeit	sind alle Texte in Deutsch und Englisch verfügbar
Funktion	Signaturerstellung	kann eine Signatur für eine Datei erstellt werden
Funktion	Signaturverifikation	kann eine Signatur überprüft werden
Funktion	Zertifikatskette verifizieren	kann ein ungültiges Zertifikat erkannt werden
Funktion	Gültigkeitszeiträume verifizieren	unterscheidet drei Modelle korrekt

Tabelle 1: Tabelle der Anforderungen an die Web-App

3.2 Entwicklungsprozess des UI

Um den Anforderungen aus Kapitel 3.1 gerecht zu werden, wurde zuerst ein Mock-Up mit dem Zeichenprogramm „Krita“[5] erstellt (siehe Abbildung 4). Als Vorlage für das Mock-Up wurde das UI des Zertifikat-Verifikation-Plug-ins aus dem JCT herangezogen (siehe Abbildung 5). Der Hauptunterschied in den beiden Designs besteht darin, dass

[5]https://krita.org/en/download/krita-desktop/

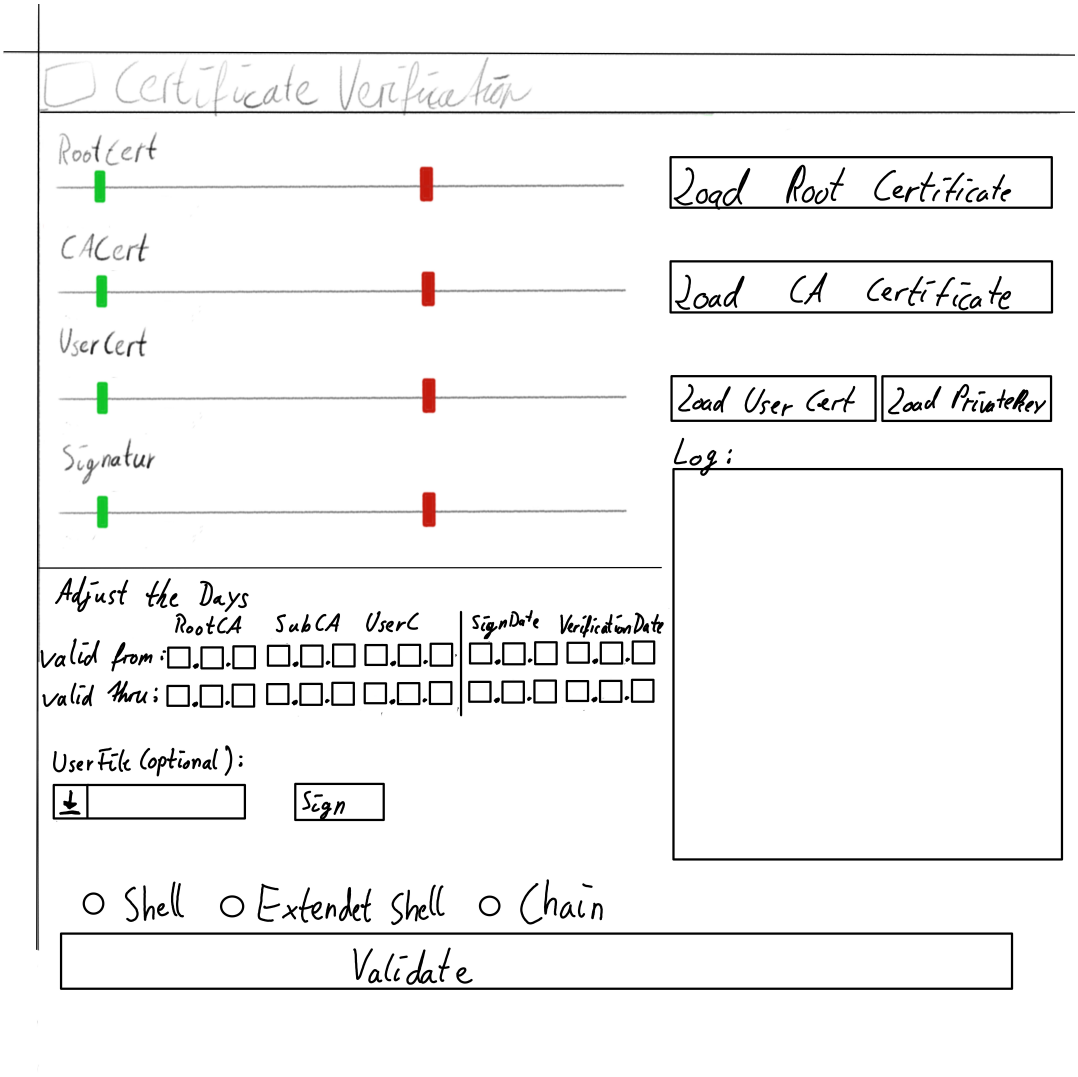

Abbildung 4: Mein erstes selbst erstelltes Mock-Up

die Daten nicht mehr über zwei separate Slider eingestellt werden, sondern über einen Range-Slider[6]. Außerdem wird das genaue Datum nicht mehr über ein Nummerfeld nur für den Tag, sondern für Tag, Monat und Jahr eingestellt.

Das anfängliche Mock-Up wurde zunächst umgesetzt. Aber die Umsetzung mit jeweils drei einzelnen Feldern für das Datum war optisch nicht ansprechend. Deshalb wurden die drei Felder durch einen einzelnen Date-Picker[7] ersetzt. Eine weitere Änderung zum anfänglichen Mock-Up ist, dass die Prüfung der Signatur und der Zertifikatskette nicht mehr auf Knopfdruck geschieht, sondern die Überprüfung durchgeführt wird, sobald sich

[6]Erlaubt das Einstellen eines Raumes durch zwei Werte.
[7]Ermöglicht sowohl ein Datum per Tastatur einzugeben, als auch ein Datum aus einem Kalender auszuwählen.

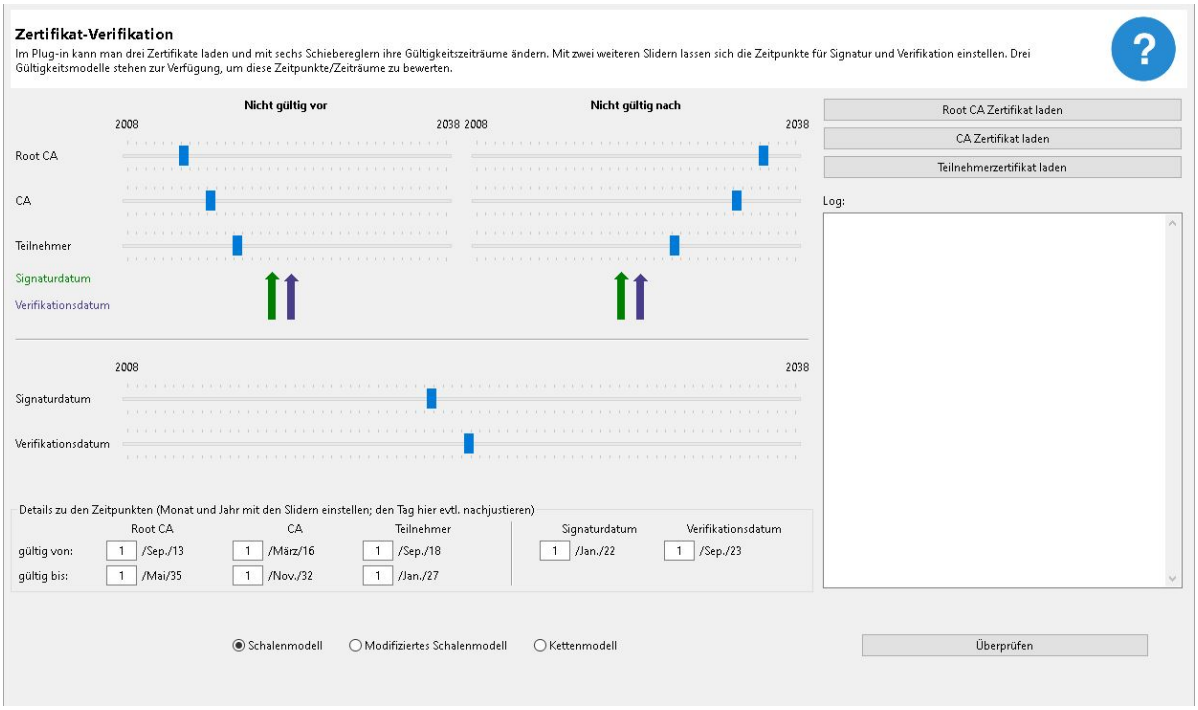

Abbildung 5: Das UI der Zertifikatsverifikation im JCT[20]

ein Parameter ändert. Hinzugekommen ist auch noch eine Liste, in der die einzelnen Schritte der Überprüfung angezeigt werden. Dies bringt den Vorteil, dass der Nutzer auf den ersten Blick sehen kann, ob und wo ein Problem vorliegt. Außerdem ist das Log zu einer einklappbaren Komponente geändert worden, weil das Log nur für Nutzer ist, die genauere Details der Analyse benötigen als in der Gültigkeits-Liste zu sehen sind. Zudem sind auch einige Komponenten optisch angepasst worden oder wurden verschoben, sodass die zusammengehörigen Teile auch zusammen stehen. Diese Summe der Änderungen ist nicht auf einmal dazugekommen, sondern stückweise über viele Iterationen hinweg bis zum jetzigen Layout (siehe Abbildung 6) hinzugefügt worden.

3.3 Beschreibung der Oberfläche

Die grafische Benutzeroberfläche wurde immer weiter an die Anforderungen aus Kapitel 3.1 angepasst, bis das jetzige Layout erreicht (siehe Abbildung 6) und in den regelmäßigen Meetings mit den Betreuern als zufriedenstellend eingeschätzt wurde. Zusammengehörige Komponenten wurden in Cards[8] gruppiert. Dadurch konnte die Übersichtlichkeit des UI verbessert werden. In der Web-App werden die folgenden 6 Cards verwendet:

[8]Die verwendete Card ist von einem Betreuer für das gesamte Projekt erstellt worden, damit die Gruppierung von Komponenten im Projekt einheitlich ist.

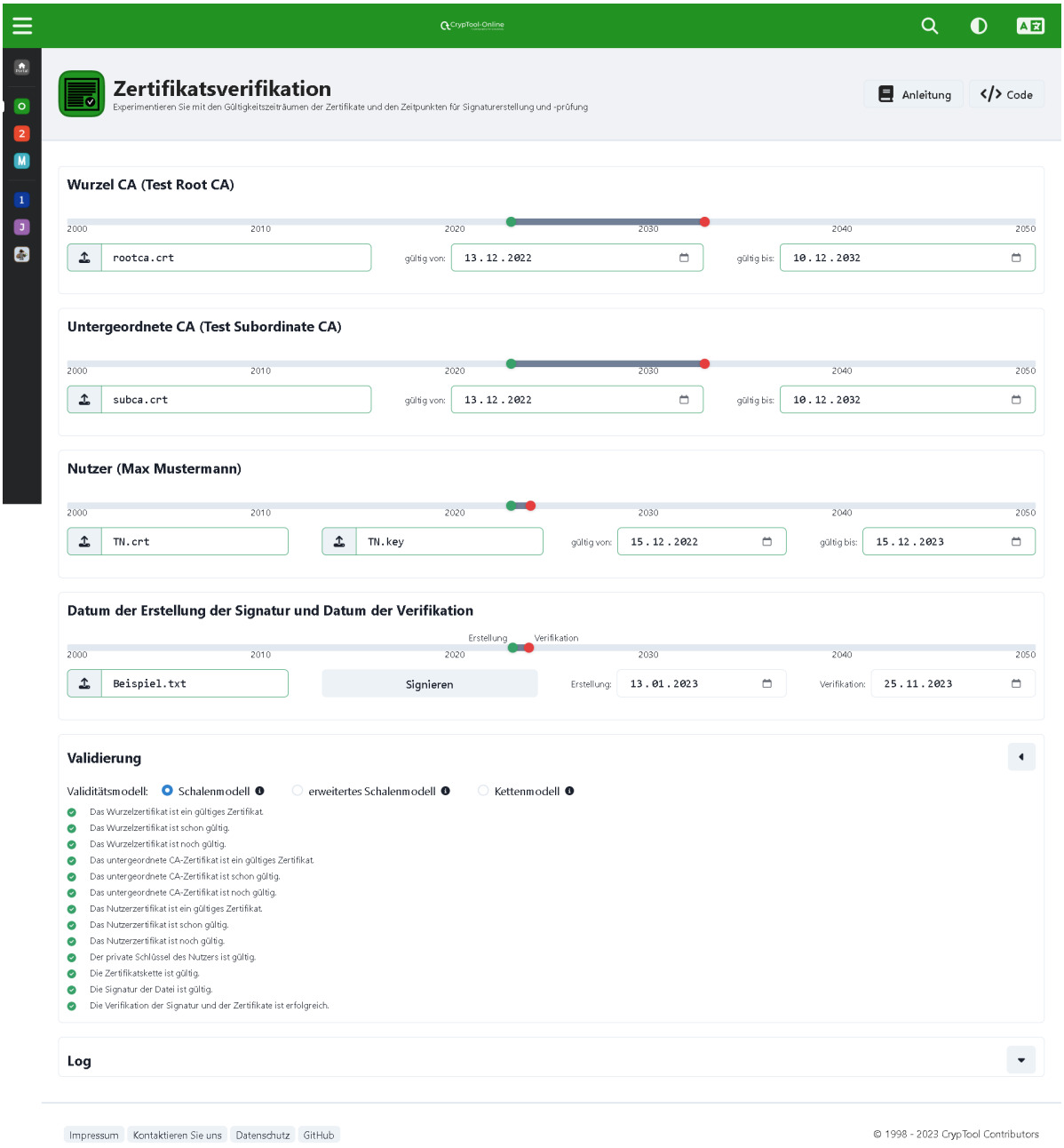

Abbildung 6: Die gesamte Website auf einen Blick

- die Wurzelcard
- die CA-Card
- die Nutzercard
- die Signaturcard
- die Validitätscard
- die Konsolencard

Wurzelcard und CA-Card sehen optisch identisch aus, sie bestehen aus einem Range Slider, einem Dateieingabefeld und zwei Datumsfeldern (siehe Abbildung 7).

Abbildung 7: Slider für Wurzel-CA und untergeordnete CA

Die Nutzercard besitzt dieselben Komponenten wie die Wurzelcard oder die CA-Card, aber zusätzlich ein weiteres Dateieingabefeld (siehe Abbildung 8).

Abbildung 8: Slider für das Nutzerzertifikat

Die Signaturcard verfügt, neben den Komponenten, die auch in der Wurzelcard vorhanden sind, über einen Knopf mit der Beschriftung „Signieren“ oder „Sign“ (siehe Abbildung 9).

Die Validitätscard beinhaltet eine Gruppe von Radio-Knöpfen, die mit den Namen der drei Gültigkeitsmodelle beschriftet sind. Darunter befindet sich eine Auflistung aller

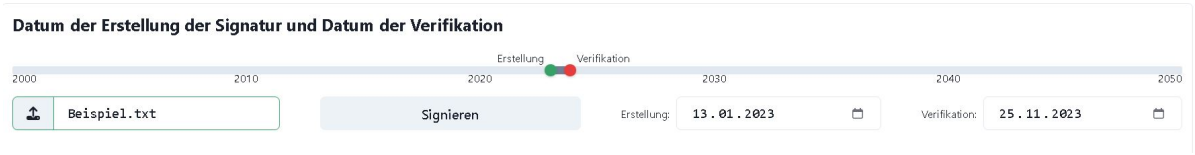

Abbildung 9: Signatur-Slider

Validierungsschritte, die mit einem Icon versehen sind, das den Status der Gültigkeit zeigt (siehe Abbildung 10).

Abbildung 10: Liste der Validierungsschritte und Modellauswahl

Die Konsolencard zeigt nur ein Textfeld, in dem das Log ausgegeben wird (siehe Abbildung 11).

Abbildung 11: Konsole

Sowohl die Konsolencard als auch die Validitätscard sind zusammenklappbar, wobei die Validitätscard standardmäßig aufgeklappt und die Konsolencard standardmäßig zusammengeklappt ist.

3.4 Funktionalität der Oberfläche

Die Oberfläche erfüllt aber nur den Zweck, den Prozess der Verifikation von Signatur und Zertifikaten für den Menschen anschaulich und interaktiv zu gestalten. Dazu muss die Oberfläche einige Funktionen haben, um die Einstellungen der Parameter für die Logik im Hintergrund verständlich zu machen. Die benötigten Funktionen sind das Einstellen von Daten und das Laden von Dateien.

Einstellung von Daten Für jedes Zertifikat können die Daten eingestellt werden, ab wann und bis wann es gültig ist. Bei der Signatur kann eingestellt werden, wann sie erstellt und überprüft wird. Das Einstellen dieser Daten funktioniert auf zwei verschiedene Weisen: Zum Einen über den Range Slider und zum Anderen über die Datumsfelder.

Die Range Slider besitzen im Code nur ganze Zahlen als Werte, die in ein Datum umgewandelt werden müssen. Dazu wird der Wert des Sliders als Monate seit dem Anfangsdatum (1.1.2000) benutzt. Im Anschluss wird berechnet, welchem Datum die Eingabe entspräche. Dieser Wert wird an der entsprechenden Stelle gespeichert. Das bedeutet, wenn der Slider auf dem Wert 120 steht, entspricht der Wert dem Datum 120 Monate nach dem 1.1.2000. Diese 120 Monate sind 10 Jahre, somit entspräche der Sliderwert 120 dem Datum 1.1.2010.

Der andere Weg ist über die Datumsfelder, die Date-Picker-Elemente von Chakra-UI [19] nutzen. Die Datumsfelder funktionieren wie ein Kalender, das heißt, es ist möglich, das Datum in einem Kalender anzuklicken oder mit der Tastatur einzugeben. So können die Daten konfiguriert werden, wobei das Datum mit dem Range-Slider auf den Monat genau und mit dem Date-Picker auf den Tag genau eingestellt werden kann. In der jetzigen Version wird das Eingabedatum des Date-Pickers erst lokal in der Komponente gespeichert, bei Fertigstellung der Eingabe auf Richtigkeit geprüft und bei Richtigkeit an entsprechender Stelle gespeichert. Bei nicht Fertigstellung der Eingabe werden die Daten verworfen (mehr dazu in Kapitel 4.8).

Laden von Dateien Das Laden von Dateien ist wichtig, um eine weitere Stufe der Interaktion zu gewährleisten. Die Dateieingabefelder sind Inputs von Chakra-UI, die so konfiguriert sind, dass sie eine Datei akzeptieren. Die Datei, die geladen wird, wird direkt bei der Eingabe zu einem JS-Objekt konvertiert. Dieses Objekt speichert den Namen der Datei, den Zeitstempel der letzten Modifikation und ein Byte Array, in dem die rohen Bytes der Datei verzeichnet sind. Es können für die verschiedenen Komponenten verschiedene Dateien geladen werden, die dann unterschiedlich verarbeitet und gespeichert werden.

Für die Zertifikate können Zertifikatsdateien (im X.509-Format als „.crt“-Datei) geladen werden (siehe Listing 1). Das Listing 1 wurde mit dem OpenSSL-Befehl „openssl x509 -in root.crt -text“ erzeugt.

```
    Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            c2:48:bf:c3:ae:a2:23:37:48:9a:95:e4:5e:91:8a:7e
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=Test Root CA
        Validity
            Not Before: Dec 13 17:36:59 2022 GMT
            Not After : Dec 10 17:36:59 2032 GMT
        Subject: CN=Test Root CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:aa:19:48:e3:51:05:80:50:57:38:06:87:1d:f3:
                    96:d5:ca:82:bb:59:87:2f:79:a2:1d:28:88:cf:4d:
                    ea:57:8c:a9:f9:50:46:21:02:66:f1:30:19:d6:3b:
                    e6:e9:a1:20:56:a1:22:6a:c4:96:c1:d0:ef:5a:7e:
                    0f:c6:46:9e:34:db:c2:7a:a1:f0:dd:aa:b0:e8:05:
                    55:07:43:58:0f:b8:57:2b:b7:03:69:97:e2:2b:aa:
                    ac:88:16:55:88:fe:6e:30:6f:7a:93:e2:ed:cb:e1:
                    7b:05:98:a8:b8:33:6d:a3:c7:d5:c1:2e:93:ea:2a:
                    03:06:e6:4f:10:02:d6:b8:fd:6a:9f:ea:33:e0:0d:
                    62:f5:0e:eb:f9:6a:84:a7:d7:8c:a6:39:43:1c:7c:
                    49:5d:b8:13:4f:cd:17:ad:0c:b3:93:2d:45:30:32:
                    d1:cc:9f:b9:7f:77:5d:5a:43:7b:7f:34:36:51:27:
                    4a:a1:78:99:a6:23:0f:c1:03:4c:bf:0d:09:45:cb:
                    a1:cd:e2:6e:db:aa:ad:71:63:c9:3b:6d:bd:55:3c:
                    5c:43:d6:08:2f:e6:a2:f9:8c:89:ca:3a:ed:d4:2a:
                    1d:bf:a0:de:5c:06:3f:4e:f0:59:e1:86:ab:c0:7e:
                    76:0f:e0:e2:63:6a:88:e7:ce:75:fa:7a:8a:fb:5f:
                    c0:91
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Subject Key Identifier:
                3E:3C:2A:90:76:49:3C:6D:92:3F:9A:84:BD:D9:2D:35:59:B4:
    FC:22
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        23:15:ec:ed:3d:97:ab:7d:7f:22:63:69:ed:04:fc:01:7c:5f:
        18:9c:00:11:23:93:bb:96:30:36:a5:a4:c2:fc:d3:a8:8a:09:
        e7:e1:05:c7:cd:3d:25:b3:36:95:64:b7:1a:e2:a2:43:95:91:
        1c:92:2d:1a:43:3f:b4:26:45:ef:ac:c2:1b:ec:ef:48:dd:92:
        43:a9:fb:8f:0b:88:8e:27:47:a3:59:43:76:2a:ec:f7:01:84:
```

```
        6e:72:d2:5b:31:43:ba:a0:c2:21:6c:0e:3c:53:e3:bf:42:9b:
        8d:f7:ee:01:cd:46:1f:27:c3:a4:19:cb:3b:b2:7f:d8:34:23:
        18:6a:68:a9:0b:b9:e3:5f:13:dd:7f:23:19:d9:60:cc:ae:c8:
        4c:24:81:6e:ba:05:e4:58:f1:b5:39:fc:c5:a6:12:d9:60:ae:
        21:3b:60:fb:5d:0b:6e:18:bf:f9:52:16:bc:c0:95:f5:40:60:
        f7:4b:c1:b9:55:e2:59:08:2e:0e:ca:46:33:29:40:23:bd:72:
        4f:b1:78:05:db:c5:24:31:b1:2a:e3:89:50:87:1d:ba:1a:ed:
        19:04:cb:64:9d:1c:29:fb:1d:08:a9:d6:07:eb:f5:57:03:93:
        a9:8f:1c:77:a1:2c:a8:16:e8:a5:e6:d7:93:c9:5a:6c:06:82:
        f0:30:3a:a5
-----BEGIN CERTIFICATE-----
MIIC+zCCAeOgAwIBAgIRAMJIv8OuoiM3SJqV5F6Rin4wDQYJKoZIhvcNAQELBQAw
FzEVMBMGA1UEAwwMVGVzdCBSb290IENBMB4XDTIyMTIxMzE3MzY1OVoXDTMyMTIx
MDE3MzY1OVowFzEVMBMGA1UEAwwMVGVzdCBSb290IENBMIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEAqhlI41EFgFBXOAaHHfOW1cqCu1mHL3miHSiIz03q
V4yp+VBGIQJm8TAZ1jvm6aEgVqEiasSWwdDvWn4PxkaeNNvCeqHw3aqw6AVVB0NY
D7hXK7cDaZfiK6qsiBZViP5uMG96k+Lty+F7BZiouDNto8fVwS6T6ioDBuZPEALW
uP1qn+oz4A1i9Q7r+WqEp9eMpjlDHHxJXbgTT80XrQyzky1FMDLRzJ+5f3ddWkN7
fzQ2USdKoXiZpiMPwQNMvw0JRcuhzeJu26qtcWPJO229VTxcQ9YIL+ai+YyJyjrt
1Codv6DeXAY/TvBZ4YarwH52D+DiY2qI5851+nqK+1/AkQIDAQABo0IwQDAPBgNV
HRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBBjAdBgNVHQ4EFgQUPjwqkHZJPG2S
P5qEvdktNVm0/CIwDQYJKoZIhvcNAQELBQADggEBACMV7OO09l6t9fyJjae0E/AF8
XxicABEjk7uWMDalpML806iKCefhBcfNPSWzNpVktxriokOVkRySLRpDP7QmRe+s
whvs70jdkkOp+48LiI4nR6NZQ3Yq7PcBhG5y0lsxQ7qgwiFsDjxT479Cm4337gHN
Rh8nw6QZyzuyf9g0IxhqaKkLueNfE91/IxnZYMyuyEwkgW66BeRY8bU5/MWmEtlg
riE7YPtdC24Yv/lSFrzAlfVAYPdLwblV4lkILg7KRjMpQCO9ck+xeAXbxSQxsSrj
iVCHHboa7RkEy2SdHCn7HQip1gfr9VcDk6mPHHehLKgW6KXm15PJWmwGgvAwOqU=
-----END CERTIFICATE-----
```

Listing 1: Mit OpenSSL interpretierter Base64-Inhalt der Datei des Wurzelzertifikats

In dieser Datei stehen alle Informationen des Zertifikats, dazu gehören:

- Die Version des Zertifikats
- Die Seriennummer, die für das Zertifikat eindeutig ist
- Welcher Signaturalgorithmus verwendet wurde
- Der common name des Ausstellerzertifikats
- Der Gültigkeitszeitraum, der als zwei Daten angegeben ist
- Der common name des Zertifikats
- Welchen Algorithmus der öffentlichen Schlüssel unterstützt
- Die Länge des öffentlichen Schlüssels

- Der Modulus des öffentlichen Schlüssels
- Der Exponent des öffentlichen Schlüssels
- Die Extensions des Zertifikats
- Die Signatur für das Zertifikat
- Das Zertifikat als Base64 codiert

Diese Informationen werden von der node-forge-Bibliothek in JS-Objekte und Attribute umgewandelt.

Aus dem Byte Array des JS-Objekts, welches über die Dateieingabe erhalten wurde, wird ein Zertifikatsobjekt generiert. Das Objekt wird erstellt, indem das Byte Array in einen String konvertiert und der String über eine Methode der node-forge-Bibliothek zu dem Zertifikatsobjekt umgewandelt wird. Das JS-Objekt hat viele Attribute, von denen aber nicht alle verwendet werden. In der Web-App werden ausschließlich die Attribute „validity“, „subject“, „publicKey“ und die „isIssuer“-Funktion verwendet. Andere Attribute wären „extensions“, „version“, „serialNumber“ und einige mehr, die für die Entwicklung dieser Web-App jedoch nicht von Relevanz sind. Es ist möglich, ein komplettes Zertifikat ausschließlich in JS zu bauen und später als Datei zu exportieren. Für die Web-App würde das aber ein unnötiges Level an Komplexität hervorrufen, weshalb darauf verzichtet wurde, diese Möglichkeit zu implementieren.

Für das Nutzerzertifikat kann der private Schlüssel geladen werden. Der private Schlüssel wird als „.key“-Datei erwartet. In diesem Format stehen sowohl der private als auch der öffentliche Schlüssel in der Datei (siehe Listing 2). Das Listing 2 wird mit dem OpenSSL-Befehl „openssl pkey -in TN.key -text“ erzeugt. Weil die .key-Datei als Base64 codiert ist, muss diese Umwandlung erfolgen, damit sie gelesen werden kann.

```
    Private-Key: (2048 bit, 2 primes)
        modulus:
            00:9a:4e:28:79:16:7a:81:87:07:4b:2f:f4:52:93:
            c2:a2:02:56:d8:56:97:b8:46:13:d9:8a:c2:9c:f5:
            a8:66:c6:a7:b7:57:0a:62:ae:aa:62:02:41:37:e3:
            00:7f:96:ae:87:01:2a:92:24:f2:bc:4f:f2:42:25:
            33:89:b9:85:44:c3:55:3b:45:fa:eb:9c:85:b5:ad:
            cd:ee:43:a2:b8:3c:fe:72:6f:7c:1d:65:eb:9e:34:
            e5:f3:e7:4a:50:1d:3a:e4:12:bb:25:c1:c4:d7:ab:
            af:98:b0:5a:a6:c5:a0:f2:c0:1c:99:b3:23:a6:eb:
            db:be:51:a6:82:02:4e:ee:80:8c:a6:26:4e:af:ed:
            39:7c:b1:3c:2a:59:ca:08:fe:51:61:d0:52:d1:e8:
            73:a4:95:84:d6:bf:12:e4:ac:bc:7f:cc:f5:78:e1:
            58:43:50:b0:0d:6b:65:f6:4a:3e:ab:6b:a8:17:7b:
            4d:e3:27:f9:44:68:b7:01:89:6f:a8:64:c9:15:d8:
            1d:88:48:a9:99:0e:86:9a:ab:af:91:18:86:b2:9d:
```

```
17              80:0c:9f:0b:96:29:51:be:58:18:a0:09:2e:1c:b4:
18              c1:ef:fe:86:dd:6e:27:73:ee:07:a4:d1:ad:c7:2b:
19              71:d1:4b:4b:f0:05:9e:b9:d4:80:6c:d2:9f:b3:06:
20              fe:c5
21          publicExponent: 65537 (0x10001)
22          privateExponent:
23              35:d8:7f:54:1d:ed:7b:04:b9:1d:5a:2b:1a:a3:d8:
24              5f:fb:8b:fd:ab:5d:ff:5e:68:ef:d9:75:a7:1b:2c:
25              7c:4c:e7:5a:d9:a3:54:a3:59:ee:a0:95:cc:a0:48:
26              dc:bd:22:c3:16:bb:99:cb:0d:ef:7b:c0:70:a7:95:
27              b0:02:4e:c8:9b:97:42:fb:5d:1f:d9:fb:68:d6:31:
28              4d:eb:49:3d:9c:7f:38:44:c2:1d:6a:23:0a:0e:b0:
29              8d:e7:d0:ad:5b:3d:e1:37:ce:19:d3:5f:d2:d8:e5:
30              0a:55:c4:14:7d:53:78:52:64:c8:97:d4:21:9c:1d:
31              8e:f0:e4:37:6b:01:cb:45:02:97:53:5f:8d:71:8a:
32              47:22:c5:dd:1d:4e:6c:f8:a8:66:c5:86:fa:1a:6a:
33              cb:94:43:4a:2f:93:3e:a3:df:e6:46:c2:06:6a:0d:
34              9a:b9:4a:d7:90:ce:f2:94:7c:a5:50:96:2a:7f:e5:
35              33:a4:86:94:39:2a:c3:f3:5d:09:e5:a9:d3:86:90:
36              1b:66:0d:eb:8d:6e:29:08:09:ba:db:b3:f2:d7:91:
37              9b:30:da:af:18:37:f5:11:8a:98:5d:4c:83:2f:3d:
38              dd:75:ca:49:5d:27:22:c0:e3:1d:6e:6e:5c:f2:b8:
39              8e:3d:fa:2a:4f:a8:07:b6:59:9f:8f:fe:86:f7:65:
40              03
41          prime1:
42              00:bb:02:25:72:52:2c:c2:52:6c:de:fa:78:48:9a:
43              9e:ee:15:89:5f:17:16:36:09:93:e8:2d:8f:1a:ed:
44              ee:c6:6f:9e:cd:b6:35:4f:1a:ec:5b:9b:ab:8e:11:
45              ad:61:2d:82:9c:d9:98:76:ec:f6:00:df:af:d8:bf:
46              27:35:2a:2b:bc:7b:6a:f0:e5:96:a6:ae:5c:f5:f4:
47              99:4b:8b:ee:03:21:ba:71:1b:aa:42:c6:99:6e:73:
48              e9:2b:79:1c:4e:2a:4e:3d:ed:e1:35:2a:fc:0b:85:
49              03:b7:61:42:f8:88:a4:cf:69:32:78:55:8b:8a:6b:
50              2c:10:38:ca:ca:db:00:35:1f
51          prime2:
52              00:d3:3b:65:74:e2:c4:ba:57:f7:3c:da:c5:79:ba:
53              02:59:ce:68:58:af:c0:6a:59:05:dd:fb:d6:e3:50:
54              6e:a5:61:91:d9:d0:a7:02:6d:57:9a:02:be:2f:bd:
55              20:78:4c:fe:ea:c7:40:7f:48:e0:72:a8:14:37:6f:
56              40:d1:fd:31:8b:fa:1f:f0:b2:ad:b2:ba:4c:5a:41:
57              fc:09:ff:e6:6a:ee:e4:be:5e:b0:df:44:34:3f:7d:
58              f9:d1:ad:74:91:b7:d9:00:4f:27:b7:2d:fe:78:54:
59              43:d9:8c:6b:18:c3:f0:a6:f9:63:72:4b:fb:36:e8:
60              c2:c3:40:8b:4a:ca:11:8b:9b
61          exponent1:
62              00:ab:6c:35:3d:4d:3c:80:48:a4:49:e5:99:5b:1c:
63              79:71:4f:46:a4:66:fa:52:40:87:57:7e:63:be:d0:
64              ba:5f:26:34:90:e7:64:88:85:ea:81:b2:fa:18:79:
65              f5:cf:ad:b3:1b:ea:8e:ca:fd:3e:27:e0:e4:45:b0:
66              3b:d5:8d:39:98:46:1d:ea:82:da:8d:22:2a:4e:bf:
67              8f:c1:e3:9c:23:3c:c8:b6:24:20:f7:a3:2b:44:47:
68              76:47:06:76:be:e4:22:e2:bb:eb:04:1c:c9:fc:19:
```

```
69              4b:25:cf:3f:5f:a9:8a:42:22:71:44:43:4e:6d:6c:
70              fe:70:44:4f:08:92:a5:e7:0b
71          exponent2:
72              42:02:4d:b2:39:2b:41:10:3c:32:c8:2a:ed:df:32:
73              dc:dd:a3:be:13:2e:0e:0d:c3:c7:9a:eb:8c:1d:96:
74              76:18:07:7c:09:b8:27:0a:04:56:1c:85:52:65:c7:
75              bf:d0:4f:d9:0e:6b:19:5b:ac:c8:27:be:29:94:9a:
76              ad:04:72:77:4b:b2:af:85:26:c2:b3:75:28:bf:76:
77              03:a9:f4:57:76:50:05:2d:92:18:33:38:20:9f:3d:
78              72:48:c2:24:c7:4f:08:c9:b3:9f:8a:bf:fa:da:5a:
79              59:a1:87:d6:35:64:fe:c8:f0:4a:c7:73:b9:b5:03:
80              28:01:03:86:5b:9f:20:01
81          coefficient:
82              5b:34:b5:50:22:03:90:0d:ea:2c:00:37:d2:1f:a7:
83              f4:65:05:90:f9:35:44:33:01:4c:93:91:e8:bb:30:
84              d5:41:87:d6:42:29:68:82:2a:e1:15:b0:7c:f6:68:
85              72:2e:0c:40:51:22:69:53:24:2f:8a:45:ec:d8:00:
86              69:68:43:c7:f6:16:c7:b4:c5:f2:dd:d6:08:14:29:
87              8a:a8:76:d6:cd:12:eb:1c:3b:7a:76:5f:60:c7:c8:
88              65:07:6e:bf:8e:b4:06:64:8e:36:d0:ad:84:fb:c1:
89              53:a8:20:82:48:17:82:16:3f:0e:5c:e5:56:22:ab:
90              f3:0a:14:27:67:5c:39:52
```

Listing 2: Mit OpenSSL interpretierter Base64-Inhalt der Schlüsseldatei

Die Datei besteht nicht nur aus den eigentlichen RSA-Schlüsseln (N,e,d), sondern auch aus den Primzahlen p,q, die zur Erzeugung der Schlüsselparameter benutzt wurden, und aus den beiden Exponenten $d \mod p-1$ und $d \mod q-1$, welche benötigt werden, um die ϕ-Funktion von N zu berechnen, und aus dem Koeffizienten, der für die Berechnung mit dem Chinesischen Restklassensatz gebraucht wird. [21]

- Der Modulus N und öffentliche Exponent e – zusammen ist das der öffentliche Schlüssel
- Der private Exponent „d“
- Die Primzahl „p“
- Die Primzahl „q“
- Den Exponent $d \mod p-1$
- Den Exponent $d \mod q-1$
- Der erste Koeffizient für den Chinesischen Restklassensatz

Der Prozess des Ladens ähnelt dem der Zertifikate. Denn auch hier wird das Byte Array der Dateieingabe in einen String konvertiert und über eine Methode der node-forge-

Bibliothek zu einem JS-Objekt umgewandelt. Das Schlüsselobjekt besitzt als Funktionen „decrypt“ und „sign“, sowie die nötigen Zahlen (wie z.B. den Modulus) als eigene Attribute. In der Web-App werden davon die „sign“ Funktion und die Attribute „n“ und „e“ verwendet. „n“ und „e“ werden benötigt, um aus einem privaten Schlüssel einen öffentlichen Schlüssel zu generieren, der für die Prüfung verwendet wird. Es wird hier geprüft, ob der private Schlüssel zu dem Nutzerzertifikat gehört.

Die letzte Möglichkeit besteht darin, in der Signaturcard eine komplett beliebige Datei zu laden. Die Datei wird dann als JS-Objekt gespeichert und das Byte Array später für die Signaturerstellung und Signaturprüfung eingesetzt. Das JS-Objekt hier ist eins zu eins das gleiche Objekt, welches durch die Dateieingabe entsteht.

Die Dateien werden in den JS-Code nahezu eins zu eins übernommen. Die zu signierende Datei wird als Byte Array eingelesen und noch mit Namen und Zeitstempel gespeichert. Bei der Schlüssel- und Zertifikatsdatei werden die entsprechenden Dateien gelesen und aus den Titeln innerhalb dieser Dateien werden einzelne Attribute eines JS-Objekts erzeugt. In den Dateien stehen also alle Informationen (siehe Listings 1 und 2), die ein Zertifikat oder Schlüssel besitzen.

3.5 Funktionen der Applikation

Neben den Funktionen für die Oberfläche (siehe Kapitel 3.4) mussten auch die Funktionen für die Verifikation implementiert werden. Damit die Verifikation von Signatur und Zertifikatskette simuliert werden kann, benötigt die Web-App die Funktionen, die in Kapitel 3.1 spezifiziert sind. Die Umsetzung der Funktionen wird in diesem Kapitel erläutert.

3.5.1 Erstellung der Signatur

Da der Prozess der Verifikation einer Signatur veranschaulicht werden soll, ist die Erstellung einer solchen Signatur mit Hilfe der Web-App Grundvoraussetzung. Die Funktion, die die Signatur erstellt, ist eine sehr kompakte Funktion, da ein Großteil der Arbeit von der node-forge-Bibliothek erledigt wird. Es werden zwei Parameter an die Funktion übergeben. Diese sind ein Dateiobjekt, wie in Kapitel 3.4 spezifiziert, und der private Nutzerschlüssel. Die Signatur wird erstellt, indem von node-forge ein Hash-Objekt erzeugt und das Byte Array der Datei im String-Format dem Hash-Objekt gegeben wird, um den „Message Digest“ zu generieren. Danach wird der „Message Digest“ mit dem privaten Nutzerschlüssel durch eine Methode des Schlüsselobjekts signiert und die fertige Signatur zurückgegeben (siehe Listing 3).

```
1    // sign file using SHA-256 and the user private key
2    export function signFile(userFile, userPrivateKey) {
3        const hash = forge.md.sha256.create()
4        hash.update(new TextDecoder().decode(userFile.bytes), "raw")
5        return userPrivateKey.sign(hash)
6    }
```

Listing 3: Die Funktion für die Signaturerstellung

In der Web-App wird der „Message Digest“ mit SHA-256 erzeugt. Das ist aber nicht der einzige Hash-Algorithmus, der von node-forge unterstützt wird. Folgende Algorithmen werden unterstützt:

- SHA-1
- SHA-256
- SHA-384
- SHA-512
- MD5
- HMAC

Es ist möglich, die anderen Algorithmen noch zu implementieren. Darauf wurde aber in dieser Arbeit verzichtet (siehe Kapitel 5.2), weil ein eigenes Menü hätte erstellt werden müssen. Von einem eigenen Menü hat der Betreuer abgeraten, weil eine Web-App am besten nur ein UI besitzen sollte.

Für die Signatur werden momentan RSA-Schlüssel verwendet, wobei node-forge auch andere Signaturalgorithmen kennt. Es wurde jedoch nur RSA implementiert, weil sonst die Komplexität für den angestrebten didaktischen Zweck zu hoch geworden wäre (siehe Kapitel 5.2).

Momentan unterstützt node-forge nur zwei Algorithmen zur Signaturerstellung: RSA und ED25519. Sollen in Zukunft andere Algorithmen verwendet werden, muss entweder eine Bibliothek gefunden werden, die diese Algorithmen implementiert und kompatibel zu node-forge ist, oder die Algorithmen müssen selbst geschrieben werden. Zum Vergleich: OpenSSL (Version 3.0.7.1) unterstützt folgende Signaturalgorithmen:

- RSA
- DSA

- ED25519
- ED448
- SM2
- ECDSA
- HMAC
- SIPHASH
- POLY1305
- CMAC

3.5.2 Verifikation der Signatur

Nachdem eine Signatur erstellt wurde, soll es logischerweise auch die Möglichkeit geben, die Signatur zu verifizieren. Eine Signatur wird überprüft, indem die Datei, die in dem „userFile“-State gespeichert ist, über ein Hash-Objekt zu einem „Message Digest“ umgewandelt wird. Danach wird über eine Methode des öffentlichen Schlüssels, bereitgestellt durch node-forge, die Signatur geprüft. Die Funktion „entschlüsselt“ die Signatur und prüft, ob das Ergebnis gleich dem „Message Digest“ der Nutzerdatei ist. Falls die beiden „Message Digests“ übereinstimmen, gibt die Methode „true“, sonst „false“ zurück (siehe Listing 4).

```
1  // verify the signature using the signature and the userfile
2  export function verifySignature(userFile, userCert, signature) {
3    const hash = forge.md.sha256.create()
4    hash.update(new TextDecoder().decode(userFile.bytes))
5    try {
6      return userCert.publicKey.verify(hash.digest().getBytes(),
       signature)
7    } catch (e) {
8      return false
9    }
10 }
```

Listing 4: Die Funktion, die die Signatur mit dem öffentlichen Schlüssel überprüft

Da die Signatur mit dem privaten Schlüssel des Nutzers erstellt wird, muss überprüft werden, ob die beiden Schlüssel auch zueinander passen. Ohne diese Prüfung kann es zu Fehlermeldungen kommen oder die Signatur würde als ungültig angezeigt, ohne dass der Fehler irgendwo im UI angezeigt würde. Dies geschieht so, dass aus dem privatem

Schlüssel ein öffentlicher Schlüssel generiert wird und dieser dann mit dem öffentlichen Schlüssel aus dem Zertifikat verglichen wird. Anschließend muss die Validität des Zertifikats geprüft werden. Der Prozess dafür ist in zwei Überprüfungen aufgeteilt, in die Verifikation der Zertifikatskette und in die Prüfung der Gültigkeitszeiträume.

3.5.3 Verifikation der Zertifikatskette

Die Zertifikatskette wird überprüft, indem die „isIssuer“-Variable der Zertifikats-Objekte verwendet wird, um die Zertifikatskette zu verifizieren. Die Überprüfung erfolgt in 3 Schritten. Zuerst wird geprüft, ob das Wurzel-Zertifikat sich selbst ausgestellt hat (self-signed). Danach muss das CA-Zertifikat vom Wurzelzertifikat ausgestellt worden sein. Letztlich muss das Nutzerzertifikat von dem CA-Zertifikat ausgestellt worden sein. Die Methode gibt ein JS-Objekt zurück, welches aus den drei Werten („valid“,„invalid“,„unchecked“) für die Zertifikate besteht. Diese Werte werden benutzt, um die entsprechenden Farben im UI zu setzten. Dabei korrespondiert „valid“ zur Farbe grün, „invalid“ zur Farbe rot und „unchecked“ zur Farbe grau.

3.5.4 Verifikation der Gültigkeitszeiträume

Die Verifikation findet über zwei Funktionen statt. Die eine Funktion prüft, ob ein Datum zwischen zwei anderen liegt – die „checkDates“-Funktion. Die andere Funktion wählt die richtigen Daten, je nach Modell, aus – die „verifyCertificateTime“-Funktion. Die „checkDates“-Funktion besteht aus mehreren „if“-Bedingungen und liefert ein Objekt, bestehend aus zwei „boolean“-Werten, die sagen, ob ein Datum nach dem einen und vor dem anderen Datum befindet. Die richtigen Daten werden anhand des Modells in der zweiten Funktion ausgewählt.

Die „verifyCertificateTime“-Funktion bekommt das Modell, die Daten des Gültigkeitszeitraums des Zertifikats, das Datum der Verifikation der Signatur, das Datum der Erstellung der Signatur und das Datum des darunterliegenden Zertifikats als Parameter. Für die drei Modelle gibt es drei Möglichkeiten, die „checkDates“-Funktion aufzurufen.

Für das Schalenmodell wird „checkDates“ mit dem Datum der Verifikation der Signatur aufgerufen. Beim erweiterten Schalenmodell wird anstatt des Datums der Verifikation das Datum der Erstellung der Signatur verwendet. Das Kettenmodell wird mit Hilfe des Datums, ab wann das darunter liegende Zertifikat gültig ist, ermittelt. Falls das Zertifikat, das geprüft wird, das Nutzerzertifikat ist, wird das Datum der Erstellung der Signatur benutzt. Für die oberen drei Fälle gilt jeweils auch, dass die Daten für den Gültigkeitszeitraum des Zertifikats mit verwendet werden.

3.6 Tabelle der erfüllten Anforderungen

Die nachfolgende Tabelle zeigt, wie Anforderungen in diesem Kapitel erfüllt oder nicht erfüllt sind.

Anforderung	Erfüllt	Bemerkungen
Nutzerfreundlichkeit	Ja	Alle zusammengehörigen Komponenten sind in einer Card
Interaktivität	Ja	Es kann jeder Parameter außer der Signatur und den Algorithmen eingestellt werden
Passend zum CTO designt	Ja	Die Oberfläche besteht aus den Cards des Betreuers und wurde von ihm als passend bewertet
Zweisprachigkeit	Ja	Vollständig
Signaturerstellung	Ja	Es kann eine Signatur mit SHA-256 und RSA erstellt werden
Signaturverifikation	Ja	Vollständig
Zertifikatskette verifizieren	Ja	Momentan über die „isIssuer“-Variable
Gültigkeitszeiträume nach drei Modellen verifizieren	Ja	Vollständig

Tabelle 2: Tabelle der erfüllten Anforderungen

4 Evaluation

In diesem Kapitel sollen die verwendeten Bibliotheken auf ihre Nutzbarkeit und ihre vereinfachenden Möglichkeiten untersucht und bewertet werden. Außerdem werden in diesem Kapitel noch einige Probleme der Entwicklung besprochen und deren Lösung vorgestellt und erklärt.

4.1 Tauglichkeit der node-forge-Bibliothek

Für die kryptografischen Funktionen musste zu Beginn der Arbeit eine JS-Bibliothek gefunden werden, weil die Funktionen selbst zu programmieren den Rahmen der Arbeit überstiegen hätte. Dazu wurden einige Bibliotheken ausgesucht,wie z.B. „OpenPGP.js“[9],„CryptoJS“[10] oder „sjcl“[11]. Von diesen kamen aber nur zwei Bibliotheken in eine engere Auswahl, weil diese überhaupt die Möglichkeit bereitstellten, eine Signatur zu erstellen. Ansonsten hätte ein Signaturalgorithmus auch noch selbst implementiert werden müssen. Diese zwei waren die Web Cryptography API[12] und die node-forge-Bibliothek [18]. Die Web Cryptography API besitzt zwar die Funktion, eine Signatur zu erstellen, aber das ist auch die einzige Funktion der Web Cryptography API, die in dieser Arbeit benutzt werden konnte. Da aber in der Web-App die Verifikation von Zertifikaten und Zertifikatsketten einen großen Teil der Funktionalität ausmachen, musste auch ein Weg gefunden werden, Zertifikate im Code zu verwenden.

Zertifikate im JS-Code zu verwenden, ermöglicht die node-forge-Bibliothek. Die Bibliothek besitzt eine Funktion, die aus einem String ein JS-Objekt erstellen kann. Dieser String muss dafür aber ein bestimmtes Format haben, da sonst ungültige Zertifikate erstellt würden: Der String muss im „PEM“-Format vorliegen, also in dem Format, das eine „.crt“-Datei hat. Das heißt, dass eine „.crt“-Datei gelesen werden und dann zu einem solchen String verarbeitet werden kann. Die Möglichkeit, eine Zertifikatsdatei zu lesen und daraus ein Objekt zu erzeugen, welches sich wie das Zertifikat nutzen lässt, war von immensem Nutzen bei der Programmierung der Web-App. Denn dadurch war es ein relativ leichtes Unterfangen, den Nutzer eigene Zertifikate in die Web-App laden zu lassen und damit zu interagieren.

Die Repräsentation der Zertifikate als JS-Objekte hat einen weiteren Vorteil. Es ist dadurch möglich, die Zertifikate dynamisch anzupassen. Das ist besonders wichtig, weil somit nicht jedes Mal, wenn ein Gültigkeitszeitraum geändert wird, ein neues Zertifikat erstellt, sondern nur das Objekt angepasst werden muss. Daraus ergibt sich leider auch

[9] https://github.com/openpgpjs/openpgpjs
[10] http://code.google.com/p/crypto-js/
[11] http://bitwiseshiftleft.github.io/sjcl/
[12] https://www.w3.org/TR/WebCryptoAPI/

ein Problem, nämlich das Problem, das die Signatur des Zertifikats natürlich durch eine Änderung ungültig wird. Weil aber bei der Überprüfung der Zertifikate das Zertifikat nicht durch Ändern des Gültigkeitszeitraum ungültig werden soll, musste eine Lösung gefunden werden.

Dieses Problem kann auch mit Hilfe der node-forge-Bibliothek gelöst werden. Das Objekt, welches das Zertifikat repräsentiert, verfügt über eine „isIssuer“-Variable, in der der Name des Austellerzertifikats gespeichert ist. Damit kann überprüft werden, ob ein Zertifikat Aussteller eines anderen Zertifikats ist. So wird die Zertifikatskette in der Web-App auf Gültigkeit geprüft. Leider ist die Lösung nicht perfekt, aber ohne den Kompromiss über die „isIssuer“-Variable zu prüfen, müssten für alle Zertifikate noch die privaten Schlüssel geladen bzw. bereitgestellt werden. Das würde den Aufwand deutlich steigern und die Übersichtlichkeit der Web-App reduzieren. Außerdem müssten die privaten Schlüssel der Zertifikate auch in einem unverschlüsselten Format vorliegen, was besonders bei selbst geladenen Zertifikaten problematisch sein kann, falls der Nutzer nicht daran gedacht hat.

Wie zu sehen ist, gibt es viele Gründe, weshalb die node-forge-Bibliothek bevorzugt wurde. Denn die Funktionen, die die Bibliothek bereitstellt, waren sehr hilfreich und sind auch sehr performant, weil node-forge in nativem JS geschrieben ist. Dadurch ist kaum eine Verzögerung zu merken, wenn ein Parameter geändert wird, was für die Nutzer sehr angenehm ist.

4.2 React

Die CTO-Website ist mit dem React Framework programmiert worden, somit wurde auch die Web-App dieser Arbeit in dem Framework geschrieben. Zu Beginn war der Aufwand enorm, sich in die Grundlagen von React einzulesen. Die Grundkonzepte davon, die Web-App in einzelne Teile zu gliedern und den Komponenten die entsprechende Funktionalität zu geben, war, ohne vorherige Erfahrung, schwierig zu lernen. Die Unerfahrenheit führte zu einigen Komplikationen am Anfang, weil die Weitergabe von Variablen und Funktionen partiell unintuitiv war. Außerdem sind durch mangelnde Erfahrung häufig Endlosschleifen entstanden, die zu Abstürzen führten.

Nach ausführlicher Recherche und Einarbeitung wurden die Prozesse von React aber deutlich klarer. Wenn React einmal verstanden wurde, war die Entwicklung mit diesem Framework sehr vorteilhaft, weil die Strukturierung des Codes sehr leicht war. Besonders die JSX-Syntax des Frameworks machte die Entwicklung angenehm, weil damit die Werte, die auf der Website in HTML verwendet werden, dynamisch mithilfe von JS aktualisiert werden können.

Grundsätzlich ist das Erlernen des React-Frameworks ein hoher Zeitaufwand, der sich aber schnell rentiert. Denn, nachdem das Verständnis für die Prinzipien des Frameworks erlangt wurde, lässt sich damit schnell ein schön strukturierter und mächtiger Code schreiben. Besonders die Übersichtlichkeit und Wiederverwendbarkeit der Komponenten sorgt dafür, dass React ein sehr gutes Framework gerade für größere Projekte ist.

4.3 Entwicklung mit Chakra-UI

Bei der Entwicklung des UI wurden React Komponenten des Chakra-UI-Frameworks genutzt. Chakra-UI ist ausgewählt worden, weil die verwendeten Komponenten kostenlos zur Verfügung stehen und die Komponenten einfach zu benutzen sind. Da die Komponenten standardmäßig schon einen Stil haben, muss der Stil nur in Ausnahmefällen angepasst werden, was, wenn nötig, über eigene Variablen sehr intuitiv gelöst ist und das Schreiben von eigenen Stilen in CSS obsolet macht. Dass die Zeit für die Entwicklung der Stile und für CSS weggefallen ist, hat den Entwicklungsaufwand für das UI drastisch reduziert, wodurch mehr Zeit in die Nutzerfreundlichkeit investiert werden konnte. Im Laufe der Entwicklung hat sich das Design mehrere Male geändert. Aber durch die vorgebauten Komponenten konnten die Änderungen schnell und unkompliziert vorgenommen werden. Denn die meisten Änderungen waren mit einfachen Variablen oder Verschiebung der Komponenten erledigt.

Ein weiterer Grund für die Verwendung von Chakra-UI war, dass mit einem Framework von Komponenten auch eine Einheitlichkeit im gesamten Projekt erreicht werden konnte. Für Nutzer der Seite wäre es nicht schön anzusehen, wenn jede Web-App ein vollkommen anderes Aussehen hätte. Weil aber jeder andere Stile bei der Programmierung und dem Schreiben von Designs in CSS hat, würden zwangsläufig alle Web-Apps unterschiedlich aussehen, wenn kein Framework benutzt würde. Außerdem existiert auch zu jeder Komponente von Chakra-UI eine Dokumentation auf der Website, die erklärt, was die Komponente kann und welche Variablen für welche Zwecke benutzt werden können.

Ein Nachteil der Entwicklung war jedoch, dass Chakra-UI ein weiteres Framework war, für welches erst Zeit in das Erlernen gesteckt werden musste. Weil erst nachgelesen werden musste, welche Variable den gewünschten Effekt auf die Komponente hat.

Alles in allem ist die Entwicklung mit Chakra-UI aber sehr positiv verlaufen, weil der Aufwand, die Dokumentation der Komponenten zu lesen deutlich schneller vonstatten ging als das Erlernen von CSS. Wenn vor Projektstart schon sehr gute CSS-Kenntnisse vorhanden gewesen wären, hätte ein zufriedenstellendes Ergebnis in einem ähnlichen Zeitaufwand erzielt werden können. Da CTO aber ein Projekt ist, welches zu großen Teilen von Studierenden mitgestaltet wird, ist die Nutzung dieses Frameworks eine sehr große Erleichterung und sichert die Einheitlichkeit auf der gesamten Seite.

4.4 Log und Gültigkeits-Liste

Das Log und die Gültigkeits-Liste sind zwei Komponenten der Web-App, in denen der User sich Status der Gültigkeitsprüfung von Signatur und Zertifikatskette ansehen kann. Bei einer einzigen Überprüfung sind die beiden Komponenten redundant, aber bei mehreren Überprüfungen kommt der Nutzen der beiden Komponenten zum Tragen. Denn die Gültigkeits-Liste zeigt nur den Status der aktuellen Prüfung, ob im Moment der Prüfung die Schritte als gültig oder ungültig evaluiert wurden. Das Log zeigt hingegen ein komplettes Log aller Überprüfungen, die bis dahin durchgeführt wurden. Ergo haben beide Komponenten ihre Berechtigung, das Log für einen detaillierten historischen Überblick, welche Änderungen zu einer Invalidierung führen; die Gültigkeits-Liste für einen schnellen Überblick, ob die momentane Konfiguration gültig ist oder nicht.

Dadurch, dass die Überprüfung der Signatur und der Zertifikatskette nicht über einen Knopfdruck gestartet wird, wie es im JCT noch der Fall war, muss das Log bei jeder Parameteränderung erstellt und ausgegeben werden. Die erste Lösung war, die entsprechenden Zeilen des Logs in der Prüfungsmethode der korrespondierenden Variablen zu schreiben. Das führte aber zu dem Problem, dass teilweise das Log unvollständig oder in falscher Reihenfolge ausgegeben wurde. Die Lösung für das Problem war es, alle Methoden aufzurufen, sobald sich ein Parameter ändert. Aus Unzufriedenheit mit der ersten Lösung wurde dann die finale Lösung implementiert. Es wurde eine neue Funktion geschrieben, die das gesamte Log ausgibt und bei jeder Parameteränderung aufgerufen wird. Das heißt, dass jetzt bei jeder Parameteränderung nicht alle Werte neu berechnet werden sondern nur alle Werte neu geprüft und ausgegeben werden.

4.5 Bewertung der Gültigkeitsmodelle

Man kann die Anwendungsfälle danach unterscheiden, ob eine Signatur innerhalb von Tagen (kurzfristig) oder von Jahren (langfristig) geprüft wird. Weil es unterschiedliche Anforderungen an die kurz- und langfristige Überprüfung gibt, haben sich auch zwei unterschiedliche Modelle durchgesetzt.

Bei kurzfristig geprüften Signaturen hat sich das Schalenmodell nach RFC 5280 [7] durchgesetzt. Das Schalenmodell ist sehr einfach zu prüfen, denn es wird nur geprüft, ob die Zertifikatskette zum prüfenden Zeitpunkt gültig ist. Der Nachteil ist, dass keine langfristige Prüfung der Signatur möglich ist, außer die Zertifikate haben sehr lange Gültigkeitszeiträume. Wenn die Signatur aber nur für einen kurzfristigen Zeitraum geplant ist, ist das kein Problem. Ein Beispiel für die Anwendung von kurzfristig zu überprüfenden Signaturen wäre das Internet. Im Internet werden solche kurzfristig zu prüfenden Signaturen benutzt, damit ein Server und ein Client sicher einen Schlüsselaustausch durchführen können. Dieser Schlüsselaustausch wird für jede Sitzung neu durchgeführt, weshalb

die Signatur für die Parameter des Schlüsselaustauschs nicht langfristig verifizierbar sein muss. Dadurch können Zertifikate mit kürzerem Gültigkeitszeitraum verwendet werden. Diese Zertifikate sorgen zwar dafür, dass häufiger neue Zertifikate ausgestellt werden müssen, bringt aber den enormen Vorteil, dass die Zeit, die Angreifer zur Verfügung haben, auch verkürzt wird. Zusammenfassend lässt sich sagen, dass das Schalenmodell sich durch seine Einfachheit und die Sicherheit für kurzfristig zu verifizierende Signaturen durchgesetzt hat.

Bei langfristig zu prüfenden Signaturen konnte sich das Schalenmodell nicht durchsetzten, weil sehr lange Gültigkeitszeiträume aller Zertifikate der Kette notwendig wären. Das gäbe Angreifern viel Zeit, die privaten Schlüssel der Zertifikate zu knacken und damit Schaden anzurichten. Daraufhin wurde mit dem deutschen Signaturgesetz das Kettenmodell entwickelt. Dieses setzte sich aber nicht durch, weil zwar das Problem mit langfristigen Signaturen gelöst wurde, aber dafür neue Probleme hinzufügte. Das größte Problem des Kettenmodells war, dass die Erstellung einer gültigen Signatur nur noch vom Nutzerzertifikat abhing. Das führte dazu, dass jeder im Besitz des privaten Schlüssels gültige Signaturen – auch über den Gültigkeitszeitraum des Zertifikats hinaus – erstellen konnte.

In Europa wurde dann das erweiterte Schalenmodell (auch Hybridmodell genannt) entwickelt. Bei diesem Modell wird geprüft, ob zum Zeitpunkt der Signaturerstellung die Zertifikatskette gültig war. Dadurch kann eine gültige Signatur ihre Gültigkeit nicht über Zeit verlieren und trotzdem kann eine gültige Signatur nur erstellt werden, solange die gesamte Kette gültig ist. Zusammenfassend kann gesagt werden, dass das erweiterte Schalenmodell der Kompromiss aus den anderen beiden Modellen ist, der die Schwächen der Modelle ausgleicht. Deshalb hat sich dieses Modell auch für langfristige Signaturen durchgesetzt, für kurzfristige Signaturen ist aber die Implementierung zu komplex, um den Aufwand zu rechtfertigen.

4.6 Heller und dunkler Modus

Die Web-App kann sowohl in einem hellen als auch in einem dunklen Layout betrachtet werden, weil viele Nutzer einen dunklen Modus bevorzugen. Um die Farbstimmigkeit in beiden Modi zu gewährleisten wurde eine eigene React Hook implementiert. Die Entwicklung der Hook war sehr einfach und funktionierte fast auf Anhieb. Die Hook basiert auf der Hook „useColorModeValue“ von Chakra-UI. Diese Hook erkennt welches Farbschema gerade ausgewählt ist und liefert dann einen Wert zurück. Der Rückgabewert ist einer der beiden Parameter den die Hook bekommt und steht einmal für den hellen und einmal für den dunklen Modus. Die selbst entwickelte Hook benutzt den Wert ob hell oder dunkel, um ein Objekt zurückzugeben in dem die verwendeten Farben spezifiziert sind. Die Hook bringt den Vorteil, dass die gesamte Farbberechnung in jeder Komponen-

te nur eine Code-Zeile in Anspruch nimmt und falls nötig auch erweitert werden könnte, um von anderen Web-Apps benutzt zu werden.

4.7 Fremde Zertifikate

Es wurde getestet, ob auch fremde Zertifikate in der Web-App verwendbar sind. Das Ergebnis dieses Tests ist positiv ausgefallen (siehe Abbildung 12). Es wurde das Zertifikat der BSI-Website[13] und dessen Zertifikatskette benutzt, wie in den common names der Zertifikate zu sehen ist. In Firefox wird die Zertifikatskette erhalten, indem in der Adresszeile das Schloss angeklickt wird. Dann wird „Verbindung sicher“ ausgewählt, danach wird „Weitere Informationen“ selektiert. Zum Schluss wird „Zertifikat anzeigen“ ausgewählt und in dem sich dann öffnenden Tab kann unter dem Punkt „Verschiedenes“ die Zertifikatskette downgeloaded werden.

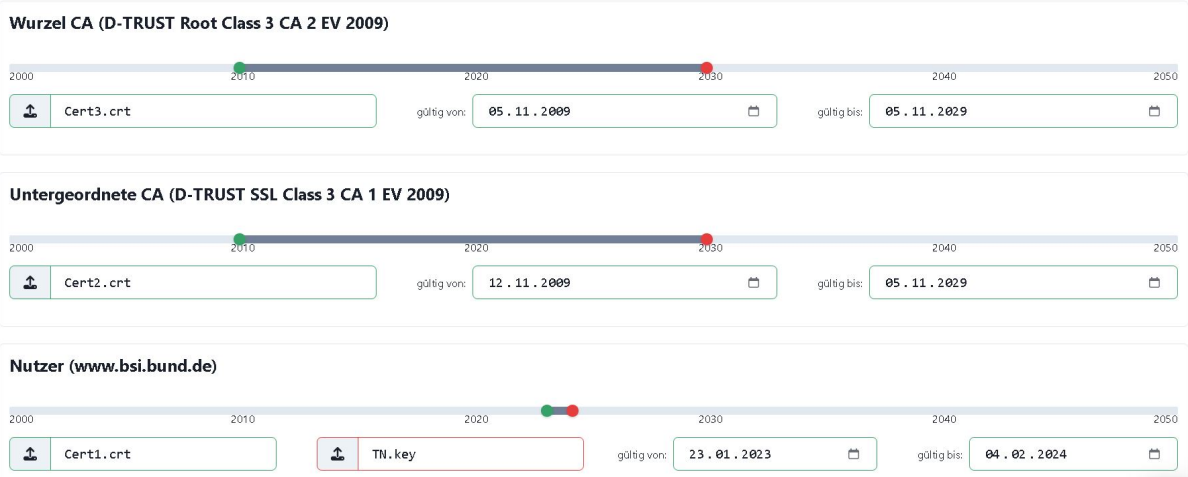

Abbildung 12: Screenshot der Web-App mit drei geladenen fremden Zertifikaten

Alle drei Rahmen um die Zertifikate sind grün. Das bedeutet, dass die gesamte Kette der Zertifikate als gültig erkannt wurde. Da eine komplette Kette verwendet wurde, ist das korrekt angezeigt. Es ist darauf zu achten, dass ein echtes Wurzelzertifikat an der ersten Stelle stehen muss, sonst wird dort direkt ein Fehler entstehen und die anderen Zertifikate werden nicht geprüft.

Als weiteres fällt auf, dass das Feld für den privaten Schlüssel rot markiert ist. Auch das ist logisch, weil der private Schlüssel der BSI-Website nur dem BSI bekannt sein sollte. Deshalb ist es nicht möglich, in der Web-App damit eine Signatur zu erstellen. Eine Prüfung einer Signatur mit dem Schlüssel des Zertifikats ist nur nicht möglich, weil

[13] https://www.bsi.bund.de/DE/Home/home_node.html

keine Signatur geladen werden kann. Wenn eine Signatureingabe hinzugefügt würde, müsste eine Signatur damit überprüfbar sein.

4.8 Weitere Kleinigkeiten in der Programmierung

Während der Programmierung sind einige Kleinigkeiten schief gegangen, die aber gefixt wurden. In diesem Abschnitt sollen diese Kleinigkeiten erwähnt und der Fix erklärt werden.

Encoding-Problem Bei Tests der Methode, die die Signatur erstellt, kam es dazu, dass beim Erstellen der Signatur einer „PDF“-Datei eine Fehlermeldung auftrat. Nach einiger Zeit der Fehlersuche ist festgestellt worden, woran das Problem lag. Der Fehler war, dass eine falsche Codierung benutzt wurde. Denn die node-forge-Bibliothek benötigt einen String, um daraus den „Message Digest“ erstellen zu können, welcher zur Erstellung der Signatur gebraucht wird.

Die Nutzerdatei liegt aber als Byte Array vor, was die Nutzung eines Text-Decoders nötig macht. Dieser Text-Decoder bekommt als Parameter das Byte Array und die Codierung, die der String bekommen soll. Bei der Codierung ist zunächst „UTF-8“ benutzt worden, da dieses aber zur Fehlermeldung führte, wurde die Codierung in „raw“ geändert. Vermutlich ist der Grund dafür, dass beispielsweise eine „PDF“-Datei Byte-Werte besitzt, die nicht korrekt in „UTF-8“ dargestellt werden können. Da die Signatur nun über die rohen Bytes erstellt wird, sollte ein solches Problem nicht mehr auftreten.

Konfiguration der Date-Picker-Elemente Die Programmierung der Date-Picker-Elemente war relativ schwierig, weil die Anforderungen sehr vielschichtig waren. Die Date-Picker können das Datum des Starts und Endes des Gültigkeitzeitraums eines Zertifikats auf den Tag genau einstellen. Allerdings soll dabei kein Datum auswählbar sein, welches nicht auf dem Slider-Element zu finden ist. Das war zunächst kein Problem, weil den Date-Pickern ein minimales und maximales Datum gegeben werden kann.

Aber dann kam es zu dem Problem, wenn das Datum über die Tastatur eingegeben wurde, konnten einige Fehler auftreten, weil ein Zwischenergebnis direkt gesetzt wurde bevor das Jahr fertig eingegeben werden konnte. Somit wäre, wenn das Jahr 2020 eingegeben werden sollte, bei der ersten Ziffer das Datum schon auf den 1.1.2000 gesetzt, was diese Eingabe unmöglich machte.

Die Lösung für dieses Problem war, dass ein weiterer React-State hinzugefügt wurde, der nur in dieser Komponente existiert. Dann wird durch ein „onBlur“-Event, also das

Fertigstellen einer Eingabe oder das Verlassen des Datumsfelds, eine Methode aufgerufen, die das Datum des lokalen States auf Richtigkeit prüft und bei Richtigkeit das Datum für das Zertifikat setzt. Ansonsten wird das letzte gültige Datum beibehalten und in dem Datumsfeld als gesetztes Datum wieder angezeigt.

Außerdem müssen die Daten für das Input-Feld in einen String umgewandelt werden, weil nur Strings im Format „JJJJ-MM-TT“ angezeigt werden können. Dazu wurde eine Funktion geschrieben, die das Datum in dieses Format umwandelt.

5 Schluss

In diesem Kapitel wird die schriftliche Arbeit zusammengefasst und die Themen werden nochmals kurz erläutert. Außerdem wird in diesem Kapitel ein Ausblick darauf gegeben, was noch möglich ist. Es können noch mehr Features hinzugefügt oder bestehende verbessert werden.

5.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Web-Applikation entwickelt, die den Verifikationsprozess einer Signatur mit den zugrundeliegenden Zertifikaten veranschaulicht. Dazu simuliert die Web-App eine Prüfung von einer Signatur und Zertifikaten, wie sie auch in einem E-Mail-Programm oder TLS ablaufen könnte.

Zur Veranschaulichung des Verifikationsprozesses wurde ein intuitives und interaktives UI entwickelt. Für die Entwicklung wurde ein Mock-Up auf Basis des UI des JCT gezeichnet und umgesetzt. In der Entwicklung wurde jedoch das Design einige Male verändert, um eine bessere Nutzbarkeit zu gewährleisten. Dazu wurden Komponenten von Chakra-UI verwendet, weil diese einfach zu benutzen sind und damit die Uniformität der verschiedenen Web-App im CTO gewährleistet ist. Dank Chakra-UI war auch die Einrichtung des dunklen Modus sehr einfach, weil dafür nur eine Chakra-UI-Hook verwendet werden muss, um die Farbwerte direkt anzupassen.

Neben dem Aussehen des UI muss die Web-App auch die Funktionen, mit denen die Prüfung der Signatur simuliert werden soll, für den Nutzer bereitstellen. Die Funktionen des UI sind das Einstellen von Daten und das Laden von Dateien.

Das Einstellen der Daten kann über einen Range-Slider oder einen Date-Picker für jedes Datum, welches den Gültigkeitszeitraum eines Zertifikates oder den Erstellungs- bzw. Verifikationszeitpunkt der Signatur bestimmt, durchgeführt werden. Der Slider dient dabei zur groben Einstellung und kann nur den Monat und das Jahr bestimmen, wohingegen der Date-Picker auf den Tag genau das Datum einstellen kann. Der Date-Picker war der komplexeste Teil der Programmierung, weil die Interaktion mit dem Slider häufig zu Problemen führte und ungültige Daten eingegeben werden konnten. Als das Problem gelöst war, funktionierten die Date-Picker nicht mehr ganz korrekt mit einer Tastatureingabe, deshalb muss bei den Date-Pickern die Eingabe zunächst gepuffert werden, bis die Eingabe vollständig ist oder abgebrochen wird. Danach wird erst das Datum in den eigentlichen State gesetzt, sofern es gültig ist, sonst wird die Eingabe verworfen.

Die React States speichern die Daten und können nur über eine eigene set-Methode gesetzt werden. Jedes Mal, wenn der State sich ändert, werden die Komponente und

deren Kinder mit den neuen Werten neu geladen. Deshalb muss die Eingabe erst in einem lokalen State gespeichert werden, damit nur der Date-Picker neu geladen wird. Sonst käme es dazu, dass falsche Daten in das Zertifikat geschrieben würden, die für Fehler sorgen.

Im jetzigen Stand wird bei einer Eingabe der lokale State geprüft, ob dieser ein gültiges Datum enthält. Falls das der Fall sein sollte, wird die set-Methode des Datums des Zertifikats aufgerufen und die Web-App komplett neu geladen. Falls nicht, wird in dem lokalen State das falsche Datum gespeichert, bis die Eingabe vollständig oder abgebrochen ist. Somit wird kein neues Laden der Web-App und keine Überprüfung ausgelöst, solange kein gültiges Datum eingegeben wurde.

Bei einer vollständigen oder abgebrochenen Eingabe, die trotzdem ungültig ist, wird die Web-App neu geladen und der letzte gültige Wert wieder in den Date-Picker gesetzt. Der State des Zertifikats befindet sich in der Hauptkomponente. Von dort aus wird das Datum an alle Kinder weitergegeben, die das Datum benötigen. Wenn dieser State geändert wird, führt das durch die React-Architektur zu einem neu Laden der Web-App, bei dem die Werte, die in den States stehen, in den anderen Komponenten gesetzt werden. Wenn also direkt der Date-Picker das Datum in den State des Zertifikats schreiben könnte, würde bei einer Eingabe ein falsches Datum geschrieben. Die Eingabe würde als vollständig angesehen – aufgrund des neu Ladens der Web-App – was eine Tastatureingabe nahezu unmöglich machte.

Dateien zu laden ist die andere Funktion des UI, damit eine weitere Ebene der Interaktion für den Nutzer bereitgestellt wird. Es ist möglich, fünf verschiedene Dateien zu laden. Drei davon sind Zertifikate, die im X.509 Format als „.crt“-Datei gefordert werden. Eine Datei ist für den privaten Schlüssel des Nutzerzertifikats vorgesehen, dieser muss im unverschlüsselten PEM-Format als „.key“-Datei vorliegen. Dateien, die nicht in den entsprechenden Formaten vorliegen, werden von der Web-App verworfen und der Nutzer darüber in Kenntnis gesetzt, welches Format gefordert ist. Die letzte Möglichkeit ist es eine komplett willkürliche Datei zu laden. Diese Datei unterliegt keinen Formatbeschränkungen, weil diese als zu signierende Datei herangezogen wird.

Die Dateien werden durch das Input-Feld in ein JS-Objekt, welches aus dem Namen der Datei, einem Zeitstempel und dem Byte Array der Datei besteht, eingelesen. Für die Zertifikate und den Schlüssel wird jeweils eine Methode von node-forge verwendet, um die Byte Arrays in Zertifikats- oder Schlüsselobjekte zu konvertieren. Bei der Nutzerdatei wird das durch das Input-Feld erstellte Objekt abgespeichert.

In der Web-App soll die Prüfung der Signatur vollständig funktionieren und nicht nur über die Daten der Zertifikate. Deshalb wurde eine echte Prüfung der Signatur und der Zertifikate durchgeführt. Dazu muss eine Signatur erstellt und verifiziert werden können sowie die Zertifikatskette und die Gültigkeitszeiträume nach drei Modellen überprüft werden.

Die Erstellung der Signatur wird mit Hilfe der node-forge-Bibliothek sehr einfach, es wird ein Hash-Objekt erzeugt, welches aus dem Byte Array der Nutzerdatei einen „Message Digest“ generiert und diesen mit dem privaten Nutzerschlüssel zu einer Signatur berechnet. Die Verifikation einer so erstellten Signatur funktioniert, indem wieder der „Message Digest“ berechnet und über den öffentlichen Schlüssel geprüft wird, ob die Signatur zu dem „Message Digest“ passt. Falls die Signatur zum „Message Digest“ passt, wird die Signatur als gültig erkannt ansonsten als ungültig.

Damit der gesamte Prozess als gültig angezeigt wird, muss auch die Zertifikatskette gültig sein. Das ist der Fall, wenn die „isIssuer“-Variable der Zertifikate auf das nächst höhere Zertifikat der Kette verweist. Bei dem obersten Zertifikat, dem Wurzel-Zertifikat, wird erwartet, dass es von sich selbst ausgestellt wurde. Die Gültigkeitszeiträume der Zertifikate werden abhängig von dem zugrundeliegenden Gültigkeitsmodell verifiziert. Dafür wird für jedes Zertifikat überprüft, ob das zu prüfende Datum zwischen den Daten liegt, die den Gültigkeitszeitraum des Zertifikates begrenzen. Die Daten, die geprüft werden, sind, für das Schalenmodell der Prüfzeitpunkt der Signatur, für das erweiterte Schalenmodell das Datum der Signaturerstellung und für das Kettenmodell der Zeitpunkt der Signatur des vorherigen Zertifikats. Beim Kettenmodell wird das Datum der Signaturerstellung überprüft, danach wird das Datum, ab wann das Nutzerzertifikat gültig ist, geprüft, darauf folgend, ab wann das untergeordnete CA-Zertifikat gültig ist.

Die drei Gültigkeitsmodelle haben unterschiedliche Anwendungsgebiete. Das Schalenmodell hat sich aufgrund der Einfachheit im Internet für Server durchgesetzt, weil dort auch keine langfristig überprüfbaren Signaturen benötigt werden. Das Kettenmodell wurde im deutschen Signaturgesetz entwickelt, konnte sich aber nicht durchsetzen, weil die Implementierung komplexer war und die Signatur nur von dem Nutzerzertifikat abhängig war. Das führte dazu, dass mit dem privaten Schlüssel auch länger als die eigentliche Gültigkeit des Zertifikats gültige Signaturen erstellt werden konnten. Für die langfristig zu prüfenden Signatur hat sich das erweiterte Schalenmodell durchgesetzt, weil die Signatur von der gesamten Zertifikatskette abhängt, aber dennoch ihre Gültigkeit nicht über Zeit verliert.

Zur Übersicht über die Gültigkeitsmodelle wurde die Gültigkeits-Liste und das Log in das UI der Web-App eingebaut. Die Gültigkeits-Liste dient zum schnellen Überblick über die aktuelle Verifikation. Für alle Nutzer, die ein genaueren Blick auf die Verifizierung werfen wollen, wurde das Log entwickelt. Im Log wird für alle Parameteränderungen die komplette Überprüfung durchgeführt und als Log ausgegeben, sodass alle Überprüfungen hintereinander angeschaut werden können.

Die Implementierung der Funktionen war nur mit einer fertigen Krypto-Bibliothek möglich, weil die Implementierung des Signaturalgorithmus und der Umsetzung der Zertifikate sehr komplex und zeitaufwändig gewesen wäre. Am besten genügte die node-forge-Bibliothek den gestellten Anforderungen. Die Bibliothek ermöglicht es, die Zertifikate als JS-Objekt zu verwenden und Signaturen zu erstellen. Das UI wurde mit Hilfe des

React-Frameworks entwickelt. Dieses führte am Anfang zu Problemen durch Unerfahrenheit. Die Probleme konnten aber durch Einlesen in die gute Online-Dokumentation und die Unterstützung der Betreuer schnell beseitigt werden.

5.2 Ausblick

Neben den jetzt implementierten Funktionen und UI-Elementen könnte die Web-App noch mit einigen Features versehen werden. Diese könnten vorhandene Funktionen erweitern oder gänzlich neue hinzufügen. Außerdem kann das UI noch ansprechender gestaltet sein oder dem Nutzer noch mehr Möglichkeiten der Interaktion bieten.

5.2.1 UI-Erweiterungen

Die Web-App verfügt momentan über ein festes Zeitintervall, in dem die Daten eingestellt werden können. Das könnte über ein Einstellungsmenü des CTO zu einem dynamischen Intervall geändert werden, falls das CTO über ein solches Menü verfügt. Damit wäre ein längerer Support in die Zukunft möglich und der Nutzer könnte auch mit älteren Zertifikaten experimentieren.

Außerdem wäre es möglich die Gültigkeits-Liste so zu erweitern, dass ein direkter Vergleich zwischen allen drei Gültigkeitsmodellen gezeigt würde. Ein Vorteil davon wäre, dass der Nutzer kein spezielles Modell auswählen müsste. Er könnte direkt für eine Situation sehen, welches Modell für diese Situation am ehesten verwendet werden soll. Ein Nachteil wäre, dass die Übersichtlichkeit verloren ginge und der Nutzer mit einem zu voll gestopften UI überfordert sein könnte. Da CTO den Anspruch hat auch auf Smartphones zu laufen, könnte man das so implementieren, dass es als weitere Darstellungsmöglichkeit nur auf Desktops angeboten wird.

Eine weitere Möglichkeit, das UI zu erweitern, wäre die Übersetzung in andere Sprachen. Weil das CTO möglichst viele Menschen ansprechen und ihnen die Kryptografie näher bringen soll, wäre die Übersetzung in weitere Sprachen neben Englisch und Deutsch von Vorteil. Allerdings werden dafür Menschen benötigt, die diese Sprachen sprechen und auch bereit wären die Wartung zu übernehmen.

5.2.2 Zusätzliche Funktionen

Die Funktion, die die Zertifikatskette überprüft, könnte so erweitert werden, dass tatsächlich die Signaturen der Zertifikate geprüft werden. Dazu müssten die privaten Schlüssel

der Zertifikate aber vorhanden sein, weil sonst keine neue Signatur erstellt werden könnte, falls ein Parameter sich ändert. Deshalb sollte das vielleicht ein optionales Feature bleiben, welches über eine Einstellung aktiviert werden kann. Denn die zusätzlichen Parameter würden das UI eventuell überladen.

Falls ein Einstellungsmenü im CTO implementiert würde, könnte man auch die Möglichkeit implementieren, andere Hash- und Signaturverfahren auszuwählen und mit diesen zu experimentieren. Im Hinblick auf die Zukunft, wäre eine solche Einstellungsmöglichkeit von Vorteil, weil die Algorithmen sich weiterentwickeln.

Außerdem könnte ein eigener Zertifikatseditor implementiert werden, in dem ein Nutzer sein eigenes Zertifikat oder sogar eine Zertifikatskette erstellen könnte, um mit dieser zu interagieren und zu testen. Der Vorteil davon wäre, dass dann so gut wie jeder Parameter vom Nutzer selbst bestimmt werden kann. Allerdings kommt dieser Vorteil mit dem Nachteil einher, dass das UI komplexer würde oder mit mehreren Unter-Menüs geregelt werden müsste. Das ist aber ein großer Aufwand, der sich eventuell nicht rentiert. Alternativ könnte das in einem Tab im OpenSSL-Plugin von CTO gemacht werden und der Output dann hier benutzt werden.

Als weiteres Feature könnte eingerichtet werden, dass eine Signatur vom Nutzer geladen werden kann. Damit könnte der Nutzer selbst erstellte Signaturen prüfen. Außerdem würde das gut mit dem Einstellen von anderen Algorithmen zusammen passen. So könnten auch gültige Signaturen mit falschen Konfigurationen geprüft werden und deshalb als ungültig ausgewertet werden.

5.3 Abschluss

Diese Arbeit hat sehr viel Zeit in Anspruch genommen. Aber zum Abschluss bin ich der Meinung, dass sich die Arbeit gelohnt hat. Ich habe sehr viel während dieser Arbeit gelernt, insbesondere was React angeht und wie eine wissenschaftliche Arbeit zu schreiben ist. Ich hatte vorher noch keine Erfahrung mit React, aber durch die Dokumentation und das Feedback der Betreuer ist eine sehr schöne und funktionale Web-App entstanden. Die Web-App wird hoffentlich vielen Nutzern helfen, in das Thema einzusteigen und ein guter Zusatz für das CTO sein. Diese Arbeit war auch das größte Projekt, an dem ich bisher beteiligt war. Für mich ergibt sich ein großer Lernerfolg für zukünftiges Projektmanagement, denn zu Beginn schien mir meine Arbeit sehr unstrukturiert zu sein, aber durch das regelmäßige Feedback ist jetzt ein gut strukturiertes Endresultat entstanden.

Literaturverzeichnis

[1] Wikipedia. *CrypTool*. 2023. URL: https://de.wikipedia.org/wiki/CrypTool (besucht am 27. 07. 2023).

[2] Wikipedia. *Digitale Signatur*. 2023. URL: https://de.wikipedia.org/wiki/Digitale_Signatur.

[3] DocuSign Inc. *Was sind digitale Signaturen?* URL: https://www.docusign.de/wie-es-funktioniert/elektronische-signatur/digitale-signatur/digitale-signatur-faq (besucht am 25. 07. 2023).

[4] SSL.com-Support-Team. *Was ist ein X.509-Zertifikat?* 2019. URL: https://www.ssl.com/de/faqs/what-is-an-x-509-certificate/.

[5] IBM. *X.509 Extensions*. 2021. URL: https://www.ibm.com/docs/en/external-auth-server/2.4.3?topic=securing-x509-extensions.

[6] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Public-Key-Infrastrukturen (PKIen)*. URL: https://www.bsi.bund.de/DE/Themen/Oeffentliche-Verwaltung/Elektronische-Identitaeten/Public-Key-Infrastrukturen/public-key-infrastrukturen.html (besucht am 13. 05. 2023).

[7] Sharon Boeyen u. a. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. 2008. URL: https://datatracker.ietf.org/doc/html/rfc5280%5C#section-3.2.

[8] Fabio Martinelli und Bart Preneel. *Public Key Infrastructures, Services and Applications*. 2009.

[9] Online Grafik-Editor. URL: https://app.diagrams.net/.

[10] Bodo Möller. „Anmerkungen zur Gültigkeit von Zertifikaten“. In: *Informatik 2007 – Informatik trifft Logistik – Band 2*. Hrsg. von Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler und Rainer Koschke. Gesellschaft für Informatik e. V., 2007, S. 179–183.

[11] Bernhard Esslinger und Dominik Schadow. *JCT-Anwenderhandbuch*. 2020.

[12] Landesbeauftragter für den Datenschutz Sachsen-Anhalt vom 01.04.2007 - 31.03.2009. *IX. Tätigkeitsbericht des Landesbeauftragten für den Datenschutz Sachsen-Anhalt vom 01.04.2007 - 31.03.2009*. 2009. URL: https://datenschutz.sachsen-anhalt.de/informationen/veroeffentlichungen/taetigkeitsberichte/tb-9/14-hinweise-zum-technischen-und-organisatorischen-datenschutz-uebersicht/145-die-elektronische-signatur-in-der-verwaltung/.

[13] Christina Vlasov. *Electronic Signature Guide*. 2022. URL: https://www.getinsign.com/blog/all-about-electronic-signature/.

[14] Europäisches Parlament und Rat. *über elektronische Identifizierung und Vertrauensdienste für elektronische Transaktionen im Binnenmarkt und zur Aufhebung der Richtlinie 1999/93/EG*. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32014R0910 (besucht am 29. 09. 2023).

[15] Wikipedia. *Elektronische Signatur*. URL: https://de.wikipedia.org/wiki/Elektronische_Signatur%5C#Abgrenzung_zur_digitalen_Signatur (besucht am 20. 06. 2023).

[16] Europäische Union. *VERORDNUNG (EU) Nr. 910/2014 DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 23. Juli 2014 über elektronische Identifizierung und Vertrauensdienste für elektronische Transaktionen im Binnenmarkt und zur Aufhebung der Richtlinie 1999/93/EG*. https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A32014R0910#d1e791-73-1. 2014.

[17] MetaOpenSource. *Describing the UI*. URL: https://react.dev/learn/describing-the-ui (besucht am 27. 07. 2023).

[18] David Lehn, Manu Sporny und Dave Longley. *Forge*. 2022. URL: https://www.npmjs.com/package/node-forge.

[19] Chakra-UI Contributors. *Chakra-UI*. URL: https://chakra-ui.com (besucht am 12. 06. 2023).

[20] JCT Community. *JavaCrypTool (JCT)*. URL: https://www.cryptool.org/de/jct/ (besucht am 03. 05. 2023).

[21] Internet Engineering Task Force. *PKCS #1: RSA Cryptography Specifications Version 2.2*. 2016. URL: https://www.rfc-editor.org/rfc/rfc8017%5C#section-3.

Abkürzungsverzeichnis

CTO	CrypTool-Online
CT1	CrypTool 1
CT2	CrypTool 2
JCT	Java-CrypTool
CA	Certificate Authority
PKI	Public-Key-Infrastruktur
CRL	Certificate-Revocation-List
JS	JavaScript
UI	User Interface
CT	CrypTool
MTC3	MysteryTwister
HTML	Hypertext Markup Language
SES	Simple Electronic Signature
AES	Advanced Electronic Signature
QES	Qualified Electronic Signature
NIST	National Institute of Standards and Technology
BSI	Bundesamt für Sicherheit in der Informationstechnik
CSS	Cascading Style Sheets

Tabellenverzeichnis

Listings

Abbildungsverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen aus dem Internet. Diejenigen Paragraphen der für mich geltenden Prüfungsordnungen, die etwaige Betrugsversuche betreffen, habe ich zur Kenntnis genommen.

Der Speicherung meiner Bachelor- bzw. Masterarbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere, dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

_____ _____

(Datum) (Unterschrift)

Inhalt des USB-Sticks

- Bachelorarbeit als PDF
- Quellcode der Applikation