Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Entwicklung eines Open Source Hash und Time basierten One-Time-Password Hardware Token

Development of an open source hash and time based one-time-password hardware security token.

Philipp Schneider

Betreuer: Prof. Dr. Konstantin Knorr

Trier, den 30. Oktober 2020

Zusammenfassung.

Um die Internetsicherheit zu verbessern setzen Internetdienst Anbieter auf Multi-Factor-Authentication Systeme, meist in Form eines Hash oder Time basierten Sicherheits-Token (HOTP / TOTP). Um mehrere dieser Token simultan in einem separaten Hardwaregerät zu verwalten, wird ein solches in dieser Ausarbeitung entwickelt. Dieses System besteht aus einer entwickelten Hardware und Software Komponente, sowie einer lokalen und entfernten Benutzerschnittstelle, welche über ein eigens entworfenes Netzwerkprotokoll kommunizieren. Um die Sicherheit der geteilten Geheimnisse zu gewährleisten wird ein besonderes Augenmerk auf die sichere Kommunikation zwischen den Schnittstellen und der Implementierung der kryptographischen Funktionen gelegt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1

Einleitung

Die Zeiten, in denen ein einzelnes Passwort ausreichend war, um seine Online-Aktivitäten zu beschützen, neigen sich dem Ende.

Immer häufiger werden Identitäten bei gesicherten elektronischen Ressourcen übernommen und missbraucht. Dies liegt unter anderem an der Verwendung von schwachen Authenticatoren seitens der Nutzer [2]. Ebenso wird dieses Problem amplifiziert dadurch, dass Nutzer ihre Authenticatoren, welche pro gesicherter elektronischen Ressource einzigartig sein sollten, wiederverwenden. Was es einem Angreifer beim Kompromittieren der Identität eines Nutzers bei einer gesicherten elektronischen Ressource häufig ebenso erlaubt, andere gesicherte elektronische Ressourcen mit der gleichen Identität ohne weiteren Aufwand ebenso zu kompromittieren. Weiterhin wird das Problem dank der unsicheren Speicherung von Authenticatoren seitens der Betreibern von elektronischen Ressourcen verschlimmert; häufige Probleme stellen dabei die Speicherung von Passwörtern im Klartext oder unter Verwendung einer trivial umkehrbaren Hashfunktion dar. Nicht ausreichend sichere, gespeicherte Authenticatoren können dazu beitragen, dass bei einem kompromittieren der gespeicherten Nutzerdaten einer gesicherten elektronischen Ressource andere gesicherte elektronische Ressourcen, bei denen der Nutzer seine Identität mit denselben Authenticatoren wiederverwendet hat, ebenso kompromittiert werden.

Um das Kompromittieren einer Identität durch das Kompromittieren eines Authenticators, wie einem Passwort, zu mitigieren wird meist ein Multi-Faktor Authenticator System eingesetzt. In einem solchen System ist ein einzelner Authenticator nicht länger ausreichend, um die Identität eines Nutzer zu bestätigen. Ebenso sollten die Authenticatoren nicht von der gleichen Authenticator Kategorie stammen; so stellen zwei Passwörter keine Multi-Faktor Authentication dar. Die Authenticator Kategorie bezieht sich dahingehend auf den Prozess, der zur Authentication genutzt wird. Authenticatoren teilen sich typischerweise in Kategorien wie: Etwas, dass sich ein Nutzer merkt (Passwort), etwas, was ein Nutzer physisch besitzt (Spezialhardware) oder biometrische Merkmale eines Benutzers (Fingerabdruck, Gesichtsstruktur).

Gliederung

Nach einem einleitenden Kapitel 1 werden dem Leser notwendige Vorkenntnisse in Kapitel 2 nahegelegt. Anschließend werden die Grundlagen der Ausarbeitung in Kapitel 3 erläutert. Nach dem Darlegen der Grundlagen wird in Kapitel 4 ein genauerer Blick auf die konzipierte Anwendung geworfen. Die generelle Verwendung der Anwendung wird in Kapitel 5 genauer erläutert. Abschließend wird eine Sicherheitsanalyse in Kapitel 6 durchgeführt; gefolgt von einer Schlussfolgerung und einem Ausblick in Kapitel 7.

1.1 Motivation

Zu den häufigsten verwendeten Authentications Mechanismen zählen Passwort, SMS oder HOTP/TOTP Security Token; wobei ein Passwort meist den primären Faktor darstellt, welcher durch einen sekundären Faktor wie SMS oder HOTP/-TOTP bestätigt wird. Durch den Anstieg der Verwendung von HOTP/TOTP Security Token stellt sich jedoch die Herausforderung, dass jeder Dienst einen eigenen Security Token benötigt. Werden diese Security Token einzeln in Hardware implementiert, so benötigt ein Nutzer oft für jeden Onlinedienst ein separates Gerät in seinem Besitz. Bei einem Software basierten Security Token ist dies nicht der Fall, da ein einzelner Software basierter Security Token mehrere Onlinedienste simultan verwalten kann. Meist wird hierzu eine App auf einem Smartphone verwendet; jedoch durch den Anstieg der Verwendung des Smartphones für alltägliche Geschäfte wie das Onlinebanking wird die Separierung der Systeme, die der Security Token darstellen sollte, aufgehoben, was die Sicherheit eines Software basierten Security Tokens auf dem gleichen Gerät beeinträchtigt.

2

Vorkenntnisse

Im Nachfolgenden werden die notwendigen Vorkenntnisse zum Verständnis dieser Ausarbeitung erläutert.

2.1 HMAC - Hash-based Message Authentication Code

Ein `Hash-based Message Authentication Code` HMAC ist eine Art `Message Authentication Code` MAC bestehend aus einer kryptographischen Hashfunktion und einem geheimen Schlüssel. Ein HMAC erlaubt es sowohl die Integrität als auch die Echtheit einer Nachricht zu verifizieren [3].

2.2 X.509 Zertifikat

Ein X.509 Zertifikat ist ein Standardformat in der Kryptographie für `Public Key`-Zertifikate. Ein solches Zertifikat enthält einen öffentlichen Schlüssel sowie eine Identität (Individuum, Organisation oder Hostname) und ist entweder von sich selbst oder von einer `Certificate Authority` (CA) in einer `Public-Key-Infrastruktur` (PKI) signiert. Bei einer PKI handelt es sich um eine hierarchische Struktur an X.509 Zertifikaten, mit einem manuell vertrauten zentralen Zertifikat, der sogenannten CA, welches das Vertrauen für alle abgeleiteten Zertifikate ausstellt. Als Hostname in einem Zertifikat kann entweder eine IP-Adresse eines Hosts oder üblicher ein `Domain Name System` (DNS) Name dienen [4].

2.3 TLS

Bei dem `Transport Layer Security` (TLS) Protokoll handelt es sich um ein auf der 5. Schicht im OSI-Schichtenmodell operierendes Protokoll. TLS sorgt dabei für die sichere Übertragung von Daten, sowie die Authentifizierung von Server und Clients in einem Computernetzwerk. Das TLS-Protokoll, aktuell in Version TLS1.3, ist der Nachfolger des `Secure Sockets Layer` (SSL) Protokolls und findet im Internet Verwendung für beispielsweise die Übertragung von Webseiten mittels des `Hypertext Transfer Protocol over TLS` (HTTPS). TLS bietet dabei aber

nicht nur die Verschlüsselung von Daten mittels einer Vielzahl an Chiffren, sondern auch die Authentifizierung von Server und optional Client mittels X.509 Zertifikaten in einer PKI, selten auch mittels eines `Pre-shared key` (PKS). Während des Verbindungsaufbaus mittels TLS muss ein Server, wie in Abbildung 2.1 dargestellt, ein von dem Client vertrautes Zertifikat vorweisen; auch kann ein Server von einem Client ein Zertifikat verlangen. Wenn Server und Client ein von dem Anderen vertrautes Zertifikat vorweisen, wird dies TLS `Mutual Authentication` genannt [5].

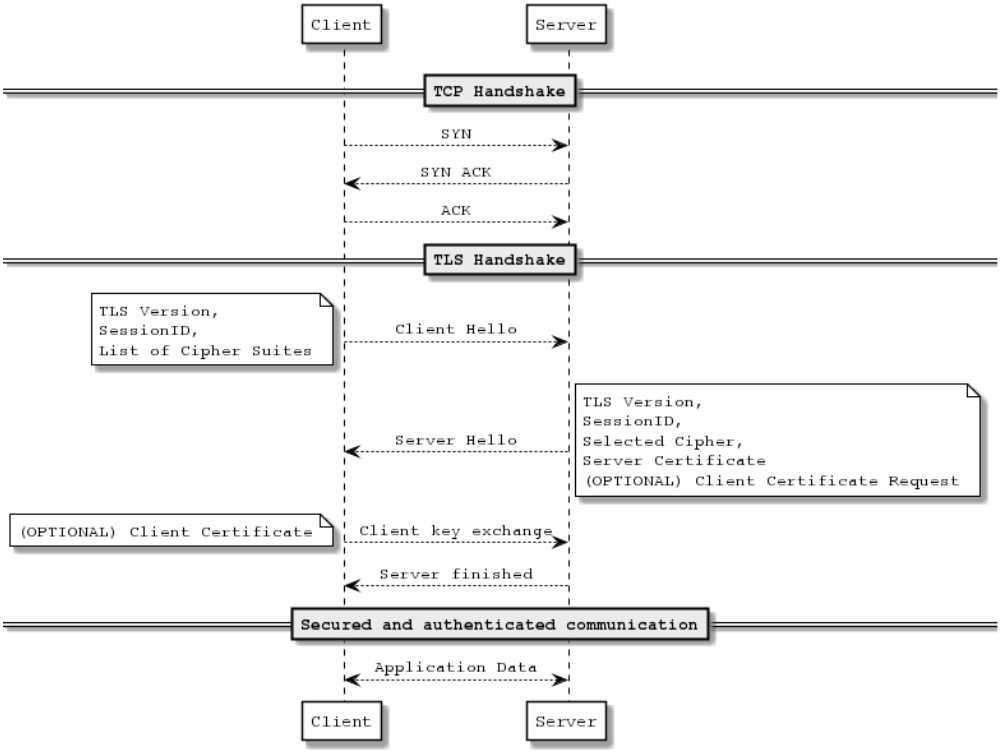

Abb. 2.1: Ablauf des TLS Verbindungsaufbaus.

2.4 Asynchrone API

Die Implementierung der Anwendung ist aufgrund der Verwendung von Netzwerkkommunikation und Hardwareinteraktion asynchron; dies spiegelt sich in der .NET Welt durch Einarbeiten einer asynchronen API in das gesamte Design wieder. Methoden, die in einer synchronen Implementierung `void` als Rückgabewert hätten, geben in einer asynchronen Implementierung `Task` zurück und erhalten nach Konvention den Namenssuffix `Async`. Methoden, die in einer synchronen Implementierung einen Rückgabewert hätten, geben in einer asynchronen Implementierung `Task<T>` wobei `T` der eigentliche Rückgabewert ist zurück; diese haben ebenso den Namenssuffix `Async`. Da asynchrone Methoden parallel laufen können aber nicht müssen, müssen alle Konsumenten mit Parallelität rechnen; dies wirkt sich

auf das allgemeine Design aus als dass alle Schnittstellen und Klassen unter Berücksichtigung von Parallelität designt werden müssen. Ebenso kann nur eine Methode, die selbst asynchron ist, eine andere asynchrone Methode aufrufen [6].

3

Grundlagen

Im nachfolgenden Kapitel werden die Grundlagen von verschiedenen Authentication Mechanismen erläutert.

3.1 Authenticator

Bei einem Authenticator ist ein Mechanismus um die Identität eines Nutzers gegenüber einem System zu bestätigen, Authentication regelt jedoch nicht ob und auf welche Ressourcen ein Nutzer zugriff erhält. Zugriffs und Ressourcenverwaltung ist Aufgabe eines Authortisationsystems und nicht die des Authenticationsystems.

3.1.1 Multi-Factor-Authentication

Bei Multi-Factor-Authentication, siehe Abbildung 3.1, ist es notwendig, dass ein Benutzer welcher sich Authentifizieren will, aufgefordert wird mehr als einen Authenticator zu verwenden. Typischerweise wird ein Authenticator bei dem der Benutzer ein Geheimnis kennt und ein Authenticator bei dem der Benutzer das Geheimnis besitzt gefordert.

3.1.2 Memorized-Secret

Bei einem Authenticator der Art "Memorized Secret"handelt es sich um einen Authentifikationsmechanismus unter Verwendung eines gemerkten Geheimnisses. Das Memorized Secret ist besser bekannt als Passwort. Die Stärke dieses Authenticators hängt von der Komplexität des gemerkten Geheimnisses ab. Dieser Authentifikationsmechanismus beweist, dass ein Benutzer das Geheimnis kennt [7].

3.1.3 Look-Up Secret

Bei einem Authenticator der Art Look-Up Secret handelt es sich um einen Authentifikationsmechanismus unter Verwendung eines Registers an vor-generierten einmal verwendbaren Passwörtern. Bei diesem Authentifikationsmechanismus wird ein Benutzer von einem Verifizierer aufgefordert, einen zufälligen Eintrag aus dem Register der vor-generierten Passwörter zu übertragen. Dieser Mechanismus beweist, das ein Benutzer im Besitz des Geheimnisses ist. Typische Beispiele für Authenticatoren dieser Art sind TAN Nummern oder Back-Up Codes [7].

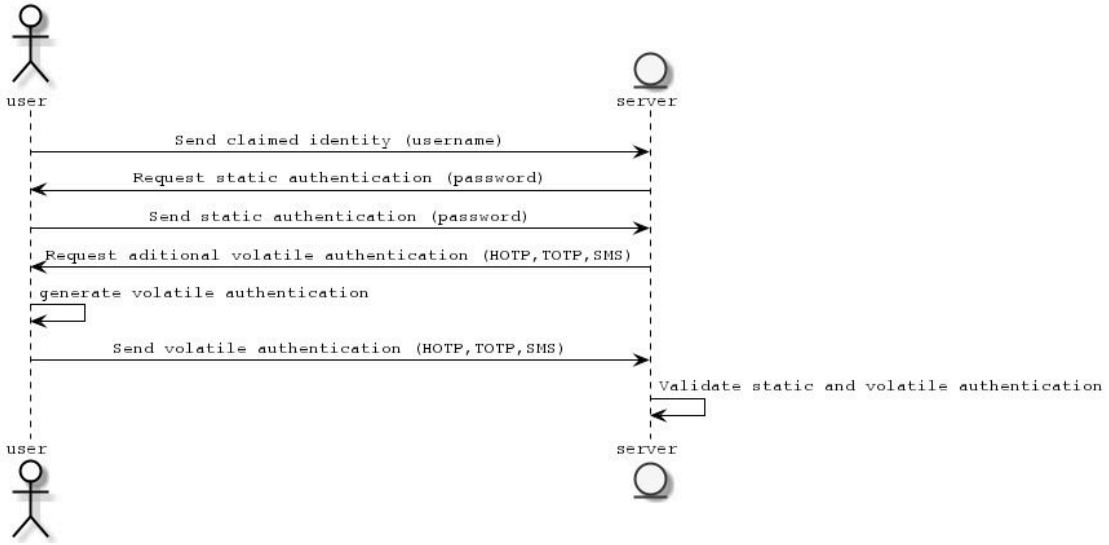

Abb. 3.1: Ablauf der Authentication unter Verwendung von Multi-Factor-Authentication

3.1.4 Out-of-Band

Bei einem Out-of-Band Authenticator handelt es sich um einen Authentifikations-Mechanismus unter Verwendung eines Gerätes, welches über einen dedizierten Kanal mit dem Verifizierer kommunizieren kann. Ein typisches Beispiel für einen Authenticator dieser Art ist ein per SMS übertragener Bestätigungscode.

3.1.5 One-Time-Password (OTP)

HOTP - HMAC-based One-time Password algorithm

HMAC-based One-time Password algorithm (HOTP) wurde entwickelt, um einen einheitlichen Standard für einen Authenticator zur Verwendung als Zweiten Faktor bereitzustellen; HOTP und abgeleitete Algorithmen sind dabei primär entworfen, um in Kombination mit einem anderen Authenticator verwendet zu werden. Der Algorithmus basiert auf einem initial über einen sicheren Kanal geteilten Geheimnis, welches dem Authenticator und dem Verifizierer zu Verfügung stehen. Nach dem initialen Austausch wird das geteilte Geheimnis nicht mehr übertragen, was den Algorithmus weniger anfällig gegenüber Man-In-The-Middle Angriffen macht. Des weiteren sind die aus dem geteilten Geheimnis generierten OTP nur einmalig gültig [8].

Funktionsweise

HOTP kann in 3 Schritten beschrieben werden, siehe Abbildung 3.2. Im ersten Schritt wird einem 20 Byte MAC mittels HMAC-SHA-1 aus dem aktuellen Zähler und dem geteilten Geheimnis erstellt. Schritt Zwei wird mittels Dynamic Truncation eine 32Bit Wert aus dem in Schritt 1 erstellten MAC erstellt. Dynamic

Truncation verwendet dabei die 4 Least Significant Bits des Least Significant Byte der Eingabe um einem Offset Index innerhalb der Eingabe zu erlangen. Das Ergebnis der Funktion sind die 4Byte an Index Offset bis Offset+3 der Eingabe. Von diesem 32Bit Wert werden nur die ersten 31Bit verwendet, um Unklarheiten mit dem Vorzeichenbit zu vermeiden. Im finalen Dritten Schritt wird der 31Bit in eine Zahl konvertiert und mittels Modulo auf einen 6 Zeichen langen Wert gekürzt. In der Abbildung 3.2 werden Schritte 2 und 3 durch den Schritt Truncate abgebildet. Das Ziel der dynamischen Kürzung ist es, den 20 Byte Wert des HMAC in einen 6 Zeichen Code zur besseren Eingabe durch einen Benutzer zu kürzen. Die Anzahl der verwendeten Zeichen ist Teil der Algorithmus Parameter; mindestens werden 6 Zeichen vorausgesetzt, funktional sind maximal 10 Zeichen möglich.

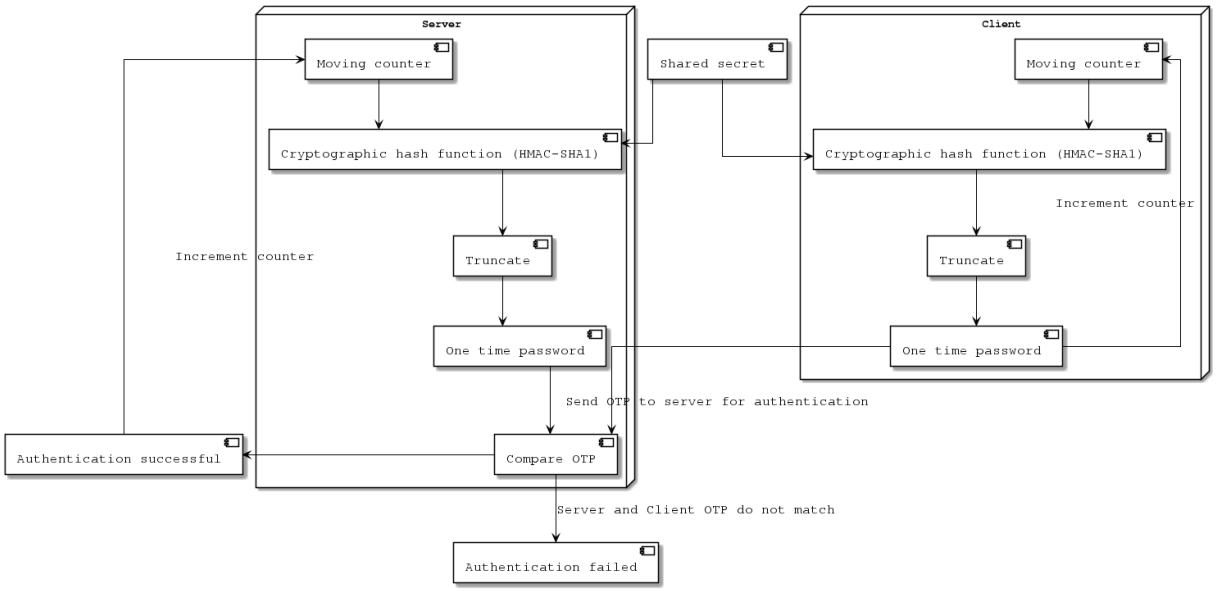

Abb. 3.2: Übersicht über die Funktionsweise von HOTP

Nach Erstellung des Codes inkrementiert der Token seinen Zähler und persistiert diesen. Auf Seiten des Verifizierers wird der Code nur bei einer erfolgreichen Authentication inkrementiert.

Probleme

HOTP hat jedoch ein Problem, dass nach mehrfachem generieren des Codes auf Tokenseite, dieser aber nicht verwendet wurde, sich Token und Verifizierer desynchronisieren. Dies tritt auf, da der Verifizierer nur bei erfolgreichen Login mit einem HOTP Code den Counter inkrementiert. Mitigiert wird dieses Problem teilweise dadurch, dass der Verifizierer eine gewisse Anzahl an Codes in der Zukunft akzeptiert. Jedoch akzeptiert der Verifizierer niemals einen Code in der Vergangenheit um Replay Angriffe zu verhindern [8].

TOTP - Time-based One-time Password algorithm

Time-based One-time Password algorithm (TOTP) ist eine Erweiterung des HOTP Algorithmus zur Unterstützung eines rollenden Zeitfaktors. Die Verwendung eines rollenden Zeitfaktors abgeleitet aus der aktuellen universellen Zeit (UTC) umgeht das primäre Problem von HOTP der Desynchronisation des geteilten Zustandes. Ebenso erweitert RFC6238 die möglichen verwendbaren Hashfunktionen in Schritt 1 um HMAC-SHA-256 und HMAC-SHA-512 [9].

Funktionsweise

Der TOTP Algorithmus ist funktional identisch, siehe Abbildung 3.3, zu HOTP mit dem einzigen Unterschied in der Berechnung des Zählers. Anstelle, das sich ein Token einen Zähler behält, wird dieser auf Seiten von Token und Verifizierer aus einem aktuellen 64-Bit Unix-Zeitstempel abgeleitet. Zur Berechnung des Zählers wird die Zeitachse seit der Unix-Epoche in 30 Sekunden große Fenster eingeteilt; der Startpunkt und die Größe der Fenster sind konfigurierbar. Zur Erstellung des Zählers wird identifiziert, in welches Fenster der aktuelle Zeitstempel zuzuordnen ist. Der vorlaufende Index des Zeitfensters wird als Zähler verwendet. Der restliche Ablauf des Algorithmus ist identisch zu HOTP mit der Ausnahme, dass nach Durchlauf der Zähler nicht inkrementiert werden muss, da dieser aus dem Zeitstempel abgeleitet und nicht gespeichert wird [**?**].

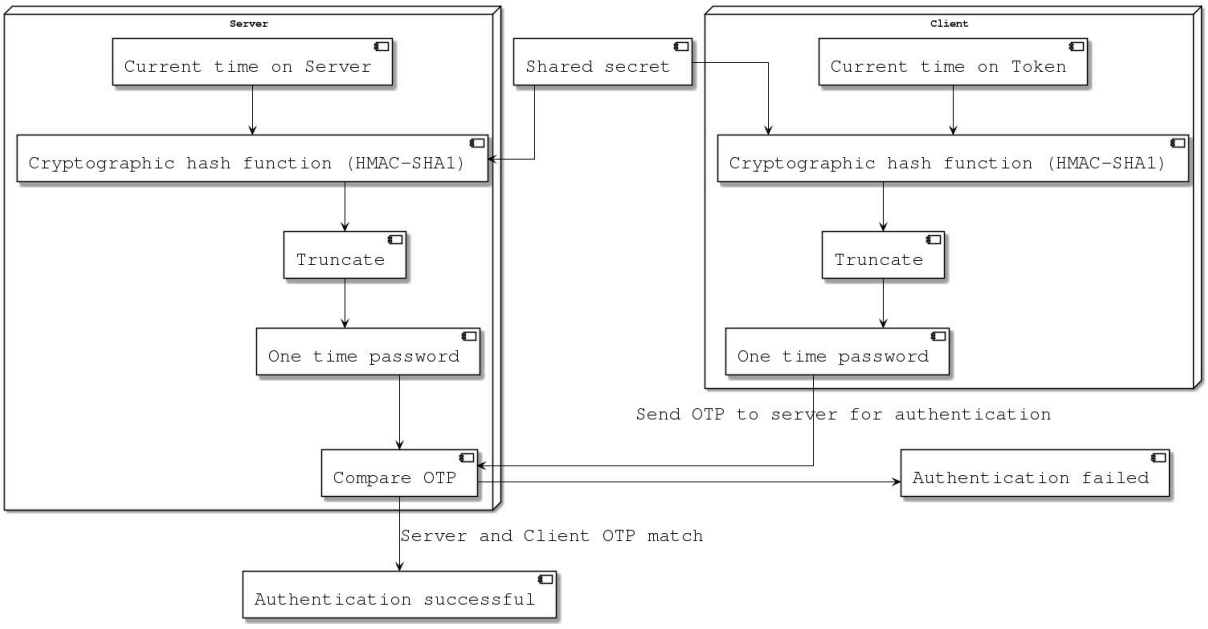

Abb. 3.3: Übersicht über die Funktionsweise von TOTP

Probleme

Das primäre Problem von TOTP ist, das sowohl Token als auch Verifizierer einen genauen aktuellen Zeitstempel benötigen. Ein Verifizierer ist meist auf einem

Server implementiert und kann die aktuelle Zeit aus dem Internet via NTP beziehen. Ein Token ist meist jedoch ein nicht mit dem Internet verbundenes System und benötigt somit eine lokale genaue Zeitquelle welche ohne externe Spannungsversorgung auskommt. Dies wird meist mittels einer Echtzeituhr mit Batterie realisiert. Die Echtzeituhr selbst fügt jedoch weitere Kosten zu der Hardware hinzu und benötigt eine hoch genüge Präzision, um über die Lebenszeit des Token die Zeit akkurat genug für die Verifizierung zu halten. Um eine gewisse Desynchronisation der Zeit auf Seiten des Token zu kompensieren, kann ein Verifizierer mehr als das aktuelle Zeitfenster akzeptieren. Dies verlängert jedoch die Validität des Codes und erhöht somit das Zeitfenster für einen Angriff auf diesen Code [9].

3.2 Security-Token

Bei einem Security-Token handelt es sich um ein Peripheriegerät, welches genutzt werden kann, um eine Identität bei gesicherten elektronischen Ressourcen zu authentifizieren. Ein Security-Token kann dabei anstelle oder zusätzlich zu einem Authenticator wie einem Passwort verwendet werden, sprich als ein Single-Factor-Authenticator oder als Teil eines Multi-Factor-Authentication Systems.

Abhängig von dem Security-Token ist dieser dabei zusätzlich mit einer Möglichkeit ausgestattet, den Anwender im Besitz des Security-Tokens zu authentifizieren. Dies kann durch Eingabe eines PINs oder vorlegen von biometrischen Daten am Security-Token geschehen und ist bei einem solchen Security-Token notwendig, damit dieser sich verwenden lässt.

Security-Token unterscheiden sich neben dem bereitgestellten Authentication-Mechanismus dahingehend ob eine logische oder physikalische Verbindung zu einem Computersystem besteht oder nicht.

3.2.1 Offline Security-Token

Bei einem Offline Security-Token besteht weder eine physikalische noch eine logische Verbindung, beispielsweise durch ein Computernetzwerk, zu einem Computersystem. Diese Art von Security-Token besitzt meist ein Nutzerinterface, wie ein Display, von dem der Authenticator von dem Nutzer manuell in ein Computersystem übertragen werden kann.

3.2.2 Online Security-Token

Bei einem Online Security-Token besteht eine physikalische oder logische Verbindung zu einem Computersystem; bei einer physikalischen Verbindung wird typischerweise eine USB-Verbindung genutzt. Eine physikalische Verbindung kann jedoch ausgenutzt werden, um den Security-Token zu beschädigen, durch beispielsweise Überspannung, oder um diesen zu kompromittieren.

Bei einem logisch verbundenen Security-Token wird die Verbindung meist durch eine kontaktlose Verbindung wie Bluetooth oder NFC realisiert. Bei logisch verbundenen Security-Tokens wird meist jedoch eine externe Spannungsquelle in Form einer Batterie benötigt.

Jedoch können Security-Token auch einen hybriden Ansatz verwenden, siehe Abbildung 3.4; eine primäre logische Verbindung und eine sekundäre physikalische Verbindung falls die logische Verbindung durch äußere Einflüsse gestört wird.

Abb. 3.4: Google Titan Security-Token Set mit USB/NFC Schnittstelle (rechts) und USB/Bluetooth Schnittstelle (links).

3.2.3 Probleme/Schwachstellen

Security-Token in Form von Peripheriegeräten stellen jedoch einen weiteren Angriffsvektor dar. Da diese im Fall des Verlustes des Security-Token, sofern nicht mit zusätzlicher Authentication im Security-Token ausgestattet, es einem Angreifer erlauben mit minimalem Aufwand eine Identität zu übernehmen.

Abhängig von dem im Security-Token implementierten Authenticator gibt es jedoch weitere Angriffe auf diesen Authenticator, diese sind jedoch meist die gleichen Angriffe wie bei einem in Software implementierten Authenticator. Sowohl bei der Hardware als auch bei der Software-Implementierung muss auf eine Side-Channel resistente Implementierung der kryptografischen Funktionen geachtet werden.

3.3 Plattform

Die konzipierte Anwendung ist angedacht auf der Hardware ohne ein dazwischenliegendes Betriebssystem direkt auf einem Mikrocontroller ausgeführt zu werden.

Nachfolgend ist die angedachte Hardwareplattform beschrieben. Die Plattform war jedoch zur Zeit der Entwicklung nicht erwerblich; als Rückfallplattform wurde eine ähnliche auf UNIX basierende Plattform der Raspberry PI verwendet. Beide Plattformen können Dotnet Anwendungen ausführen und verfügen über eine API zum Zugriff auf die unterliegende Hardware.

3.3.1 MEADOW

Die Hardwareplattform Meadow, siehe Abbildung 3.5, von Wilderness Labs ist ein Mikrocontroller mit einer eingebetteten Dotnet Laufzeitumgebung. Dies ermöglicht es diesem auf dem STM32F7 Mikrocontroller Dotnet Anwendungen welche für eine Desktopumgebung entwickelt wurden direkt auszuführen. Ebenso bietet die Plattform direkten Zugriff auf die unterliegende Hardware was es ermöglicht, angeschlossene Peripheriegeräte wie LCD Displays und Knöpfe anzusteuern. Onboard Funkschnittstellen werden durch den ESP32 Co-Prozessor zur Verfügung gestellt.

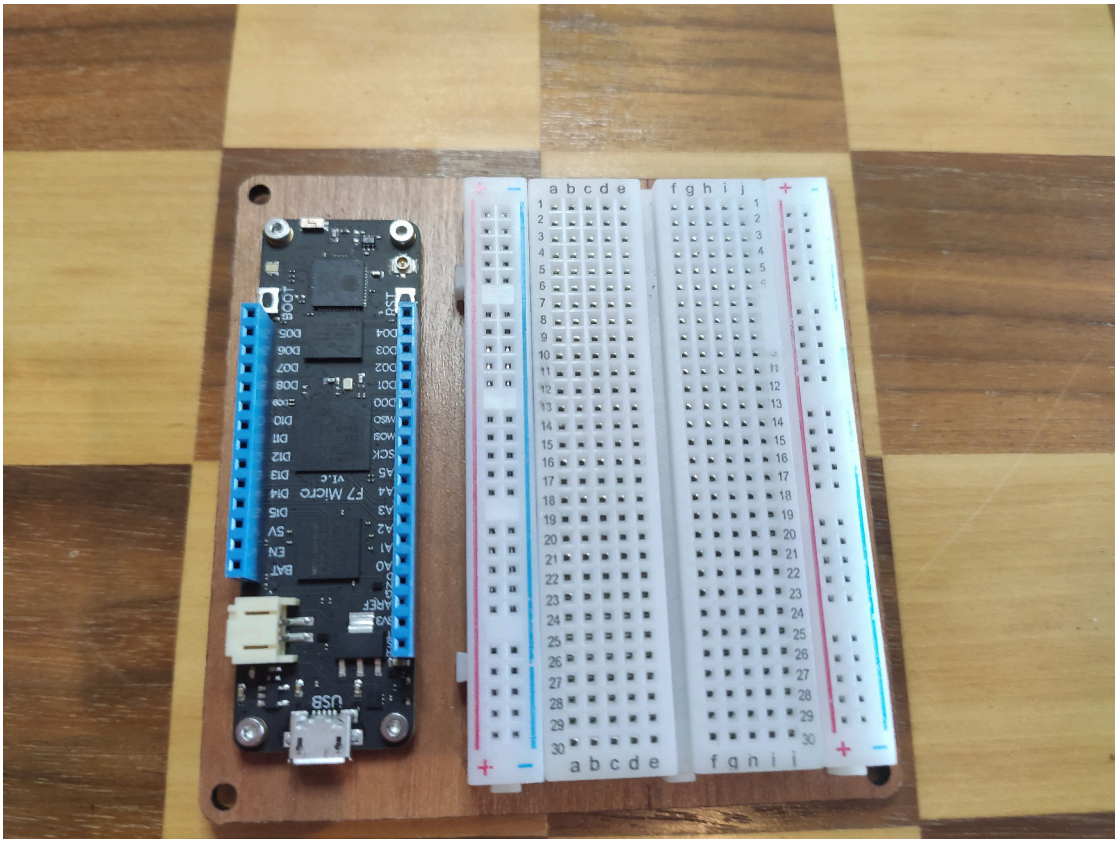

Abb. 3.5: Die Meadow Plattform von Wilderness Labs

3.3.2 Raspberry PI

Die Hardwareplattform Raspberry Pi, siehe Abbildung 3.6, von der Raspberry Pi Foundation ist ein auf ARM basierender Einplatinenrechner mit Support für Linux. Der Raspberry Pi unterscheidet sich von einem gewöhnlichen Desktopsystem primär in seinem kleinen Formfaktor sowie der Fähigkeit, auf die Hardware direkt zuzugreifen. In primärer Form geschieht dies durch den Zugriff auf die GPIO Pins des Raspberry Pis. Auf diese kann entweder durch eine API im Linuxkernel oder durch ein spezielles Arbeitsspeicher basiertes Dateisystem zugegriffen werden. Durch die Bereitstellung der GPIO Pins durch das Arbeitsspeicher basierende Dateisystem ist es nicht notwendig, auf die native API des Linuxkernels zuzugreifen was der konzipierten Anwendung ermöglicht, nur mit managed Code auf die Hardware zuzugreifen.

Die für ARM kompilierte Dotnet Laufzeitumgebung wird von Microsoft für den Raspberry PI bereitgestellt. Aufgrund dessen, das portable Dotnet Binaries auf einer beliebigen Laufzeitumgebung ausgeführt werden können, eignet sich der Raspberry Pi als eine alternative Hardwareplattform zum Betreiben der konzipierten Anwendung

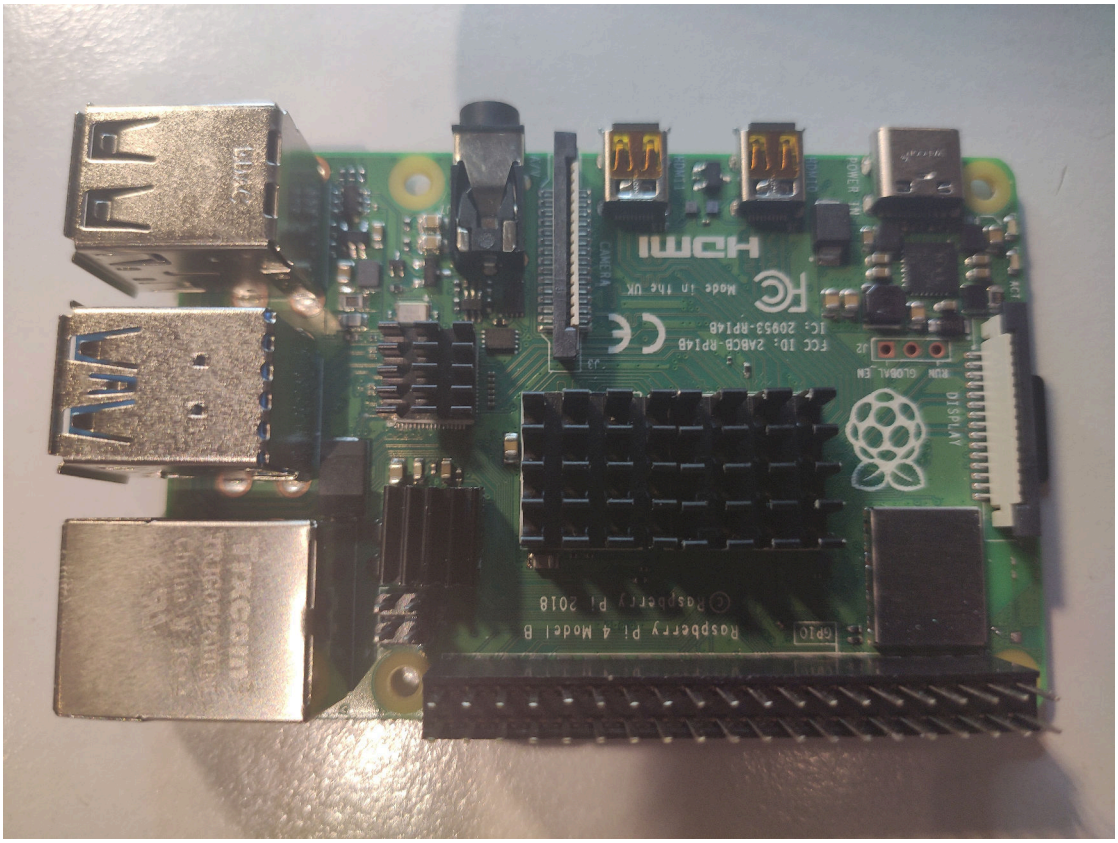

Abb. 3.6: Ein Raspberry Pi B 4 von der Raspberry Pi Foundation

3.4 Technologie

3.4.1 Blazor

Blazor ist eine von Microsoft veröffentlichte experimentelle GUI Bibliothek zum Erstellen von clientseitigen UIs in einem Browser. Die Blazor Bibliothek ermöglicht es, CSharp Quellcode zwischen Server und Clients in Webbrowsern zu teilen. Blazor benutzt JavaScript, um CSharp Code mit dem Document Object Model (DOM) des Browsers interagieren zu lassen.

Serverseitige Operation

Das Framework kann in zwei verschiedenen Modi betrieben werden: clientseitig und serverseitig. Im serverseitigen, siehe Abbildung 3.7, Modus verhält sich Blazor ähnlich zu bestehenden UI Bibliotheken wie ASP.NET Core insofern, dass das Rendern des UIs auf dem Client stattfindet währenddessen komplexere Aufgaben auf dem Server durchgeführt werden. Dabei kommunizieren Client und Server über eine SignalR-Verbindung miteinander.

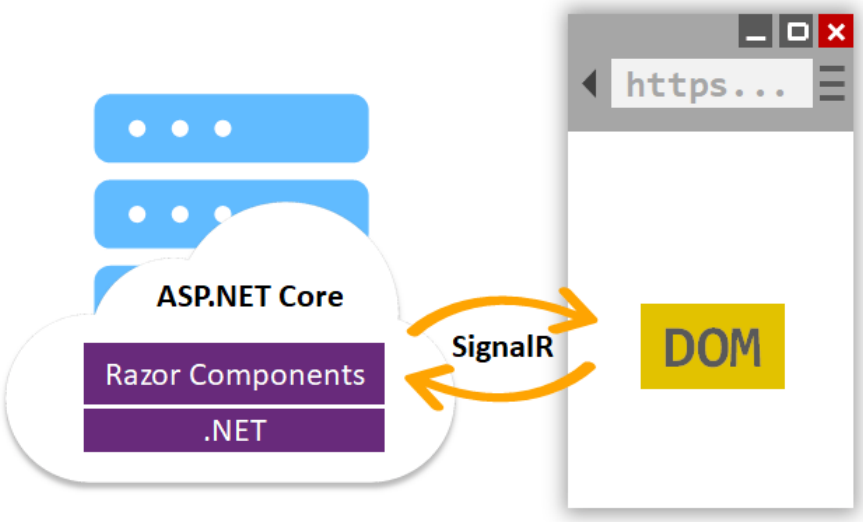

Abb. 3.7: Blazor Serverseitige Operation (Quelle: Introduction to ASP.NET Core Blazor von Microsoft. Lizenziert für nicht-kommerzielle Nutzung[1])

Clientseitige Operation

Alternativ kann Blazor komplett im clientseitigen Modus, siehe Abbildung 3.8, arbeiten. In diesem Modus wird die Dotnet Laufzeitumgebung vom Client in Webassembly ausgeführt. Webassembly ist eine ergänzende Technologie zu JavaScript, welche im Gegensatz zu JavaScript keinen direkten Zugriff auf das DOM hat, dafür jedoch keine interpretierte Sprache ist sondern ein im voraus kompilierter

Bytecode, was es Webassembly erlaubt, für rechen intensivere Aufgaben zu verwenden. Dies ermöglicht es, dass keine Berechnung oder Daten auf dem Server durchgeführt oder gehalten werden. In diesem Modus wird der Server lediglich als Bereitstellungsmedium für den auszuführenden Quellcode genutzt und nicht für Berechnungen von Clientdaten.

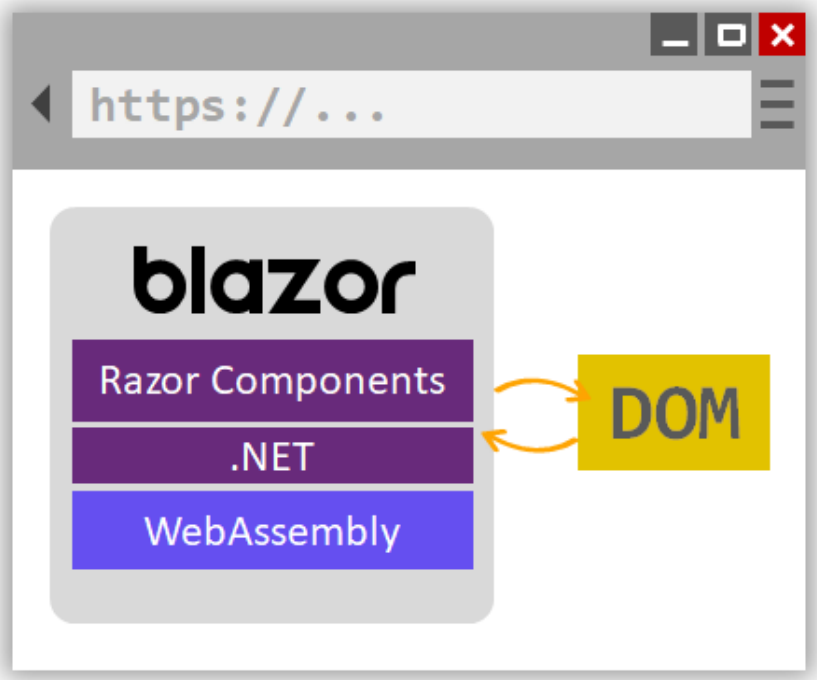

Abb. 3.8: Blazor Clientseitige Operation (Quelle: Introduction to ASP.NET Core Blazor von Microsoft. Lizenziert für nicht-kommerzielle Nutzung[1])

Dies ermöglicht weiterhin, das ein Client, sobald dieser einmal die Anwendung geladen hat, ohne Verbindung zum bereitstellenden Server weiterarbeiten kann.

3.4.2 Dotnet

Zur Implementierung des Prototypen wird auf die Dotnet Umgebung, genauer die Sprache CSharp, gesetzt. Dies bietet den Vorteil, das Code vor allem Netzwerkcode zwischen der Hardware und dem Client ohne Neukompilierung wiederverwendet werden kann. Dies ist, weil sowohl die primäre Plattform, Meadow, als auch die sekundäre Plattform, Raspberry Pi, eine vollwertige Netstandard 2.1 kompatible Dotnet Laufzeitumgebung zur Verfügung stellen. Ebenso ist diese Laufzeitumgebung auch auf Clientseite im Webclient mittels Webassembly verfügbar.

4

Entwurf

So wird also ein Hardware basierter Security Token benötigt, welcher vergleichbare Eigenschaften, in Bezug auf die simultane Verwaltung, wie ein auf Software basierender Security Token aufweist.

4.1 Anforderungen

An die konzipierte Anwendung, mit dem vorläufigen Namen Multiple One Time Pad Management (MOTP Management), soll die aufgelisteten technischen Anforderungen sowie Anwendungsfälle erfüllen:

- Req. 1: Der konzipierte Token muss über eine lokale Benutzerschnittstelle zum Benutzen des Gerätes verfügen.
- Req. 2: Für die Funktion von TOTP benötigt der Token zu jedem Zeitpunkt einen akkuraten Zeitstempel der aktuellen Zeit.
- Req. 3: Für die Funktion von HOTP benötigt der Token einen persistenten Speicher.
- Req. 4: Der Token benötigt die Fähigkeit pro Onlinedienst die verschiedenen Parameter der HOTP/TOTP Algorithmen zu berücksichtigen.
- Req. 5: Um einen Verlust bei Hardwarefehler zu vermeiden muss der konzipierte Token in der Lage sein, seine gespeicherte Konfiguration sicher auf einen anderen Datenträger zu sichern.
- Req. 6: Um die Benutzerfreundlichkeit in Bezug auf die Bedienung zu verbessern, soll ein Benutzer zusätzlich in der Lage sein, den konzipierten Token mittels einer einfachen Weboberfläche zu bedienen.
- Req. 7: Um die Sicherheit zu gewährleisten darf die Weboberfläche keinen Zugriff auf gespeicherte Schlüssel oder daraus abgeleiteten Werten erhalten.
- Req. 8: Die Verbindung zwischen der Weboberfläche und dem konzipierten Token muss beidseitig gesichert und authentifiziert sein.
- Req. 9: Um die Funktionsfähigkeit der kryptografischen Funktionen zu gewährleisten werden diese durch Unit-Tests kontrolliert.

Ein Benutzer kann über zwei verschiedene Schnittstellen mit der konzipierten Anwendung interagieren. Zum einem kann der Nutzer direkt über eine lokale

Schnittstelle mit dem System interagieren. Diese lokale Schnittstelle verfügt über volles Vertrauen zum verwaltenden Zugriff auf das Kernsystem. Über diese lokale Schnittstelle kann ein Benutzer das Anzeigen, Verwalten (Editieren vorhandener Parameter) von Token steuern, siehe Abbildung 4.1. Des weiteren kann über die lokale Schnittstelle der Zugriff der entfernten Schnittstelle zum System gewährt werden. Über die entfernte Schnittstelle ist ein Benutzer in der Lage, Token zu entfernen oder neu anzulegen. Ebenso kann der Nutzer das Anzeigen eines Token auf der lokalen Schnittstelle veranlassen. Die entfernte Schnittstelle erhält aus Sicherheitsgründen nach dem einmaligen Hinzufügen eines Token keinen Zugriff mehr auf dessen Parameter.

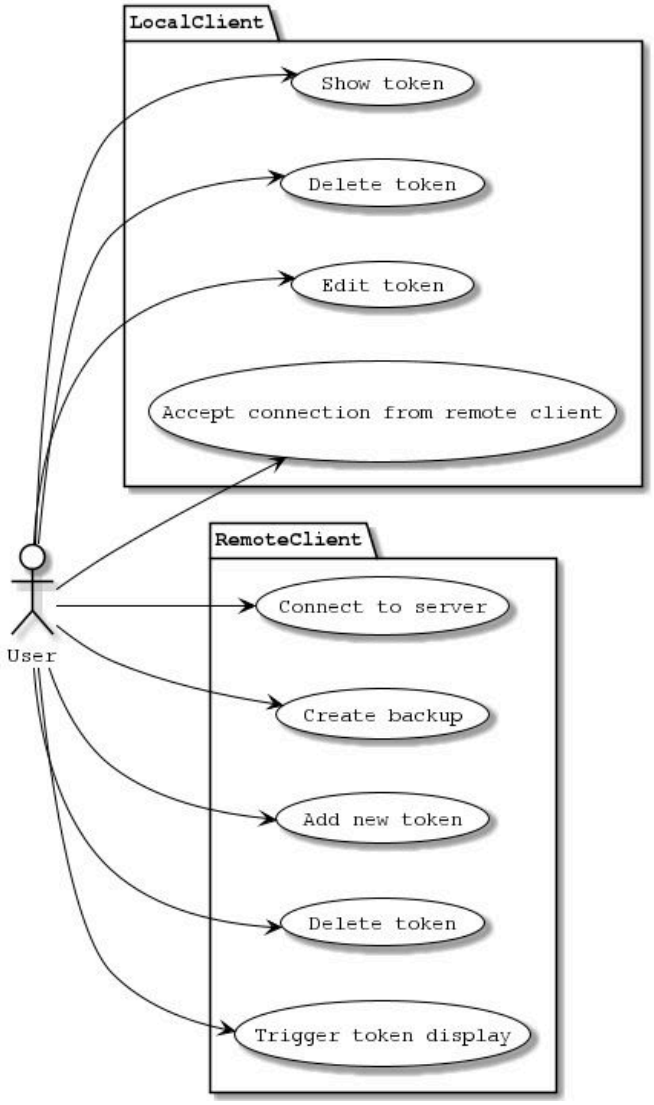

Abb. 4.1: Die Anwendungsfälle des Nutzers und die konzipierte Anwendung.

4.2 Display

Bei dem für die konzipierte Anwendung verwendeten Display handelt es sich um ein 4 Zeilen mal 20 Spalten Liquid Crystal Display (LCD) basierend auf dem HD44780U LCD Controller von Hitachi, siehe Abbildung 4.2. Der HD44780U LCD Controller von Hitachi ist ein alphanumerischer Dot-Matrix LCD Controller, entwickelt im Jahr 1985. Der Controller unterstützt verschiedene Layouts von Zeilen und Spalten, von 1x16 über 2x20 zu 4x20. Zur Kommunikation mit einem Hostsystem verwendet der HD44780 einen 4Bit bis 8Bit parallelen extern getakteten Bus. Der HD44780U verfügt über drei Register einem 8Bit Steuerregister, einem 80Byte Datenregister und einem 64Byte Zeichengeneratorregister. Über einen externen Pin wird gesteuert, ob der parallele Datenbus in das Steuerregister oder Datenregister schreibt. Gesteuert wird der HD44780U durch Schreiben von 8Bit Steuerwörtern in den Steuerspeicher [10].

Abb. 4.2: Übersicht über eine Auswahl des Zeichensatzes von einem HD44780U basiertem LCD auf einem 4x20 LCD.

Das Datenregister, siehe Abbildung 4.3, wird durch schreiben von Bytes über den parallelen BUS im Datenmodus sequentiell beschrieben. Mithilfe des Steuerregisters kann die Position des internen Index des Datenregister gesetzt werden.

Die Zuordnung zwischen Datenregister und der Zeile und Spalte des anzuzeigenden Zeichens ist abhängig von dem Layout des Displays.

Zusätzlich kann das Zeichengeneratorregister mit 4-8 Custom Sprites, abhängig von der Sprite Größe, beschrieben werden.

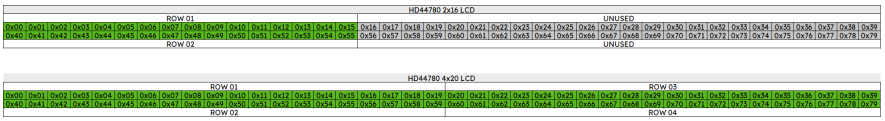

Abb. 4.3: Übersicht über das Datenregister eines HD44780 und Zuweisung von verschiedenen Layouts zu Speicheradressen.

4.2.1 Architektur

Zum Ansteuern des LCD wird aus Sicht der Software ein Treiber benötigt, welcher das Protokoll des HD44780 unterstützt. Dieser Treiber kommuniziert über die IConnection Schnittstelle mit den Registern. Die IConnection Schnittstelle wird primär durch einen vollen 8Bit Bus oder einen halben doppelt getakteten 4Bit Bus implementiert.

4.2.2 GPIO Bibliothek

Um den Treiber Plattform unabhängig zu halten interagiert dieser mit den Datenpins welche den parallelen Bus anhand einer Schnittstelle, siehe Abbildung 4.5. Die Schnittstelle `IDigitalPin` stellt einen digitalen General Purpose Input Output (GPIO) dar. Auf der Raspberry Pi Plattform wird die Schnittstelle durch eine auf Sysfs basierende Bibliothek bereitgestellt, siehe Abbildung 4.6.

Mittels Sysfs können digitale Pins durch schreiben in spezielle Systemdateien exportiert werden. Nachdem ein Pin exportiert wurde, werden vom Linuxkernel spezielle Dateien innerhalb von Sysfs angelegt. Diese Dateien können genutzt werden, um die Input / Output Richtung des digitalen Pins zu kontrollieren. Ebenso kann der Wert des Pins durch lesen oder schreiben einer Datei eingelesen oder geschrieben werden.

Auf der Meadow Plattform werden die digitalen Pins durch eine in der Firmware eingebetteten Bibliothek bereitgestellt.

4.3 Grafikbibliothek

Durch die Display Bibliothek wird ein reines Zeichen basiertes Ausgabegerät abgebildet. Um dieses effektiver zu nutzen wird eine Bibliothek konzipiert, welche simple wieder verwendbare UI Elemente über einem LCD bereitstellt. Diese Bibliothek integriert ein 5-Knöpfe D-Pad (Links, Rechts, Hoch, Runter und Eingabe) mit den UI Elementen um die Interaktion eines Nutzer mit diesen zu ermöglichen.

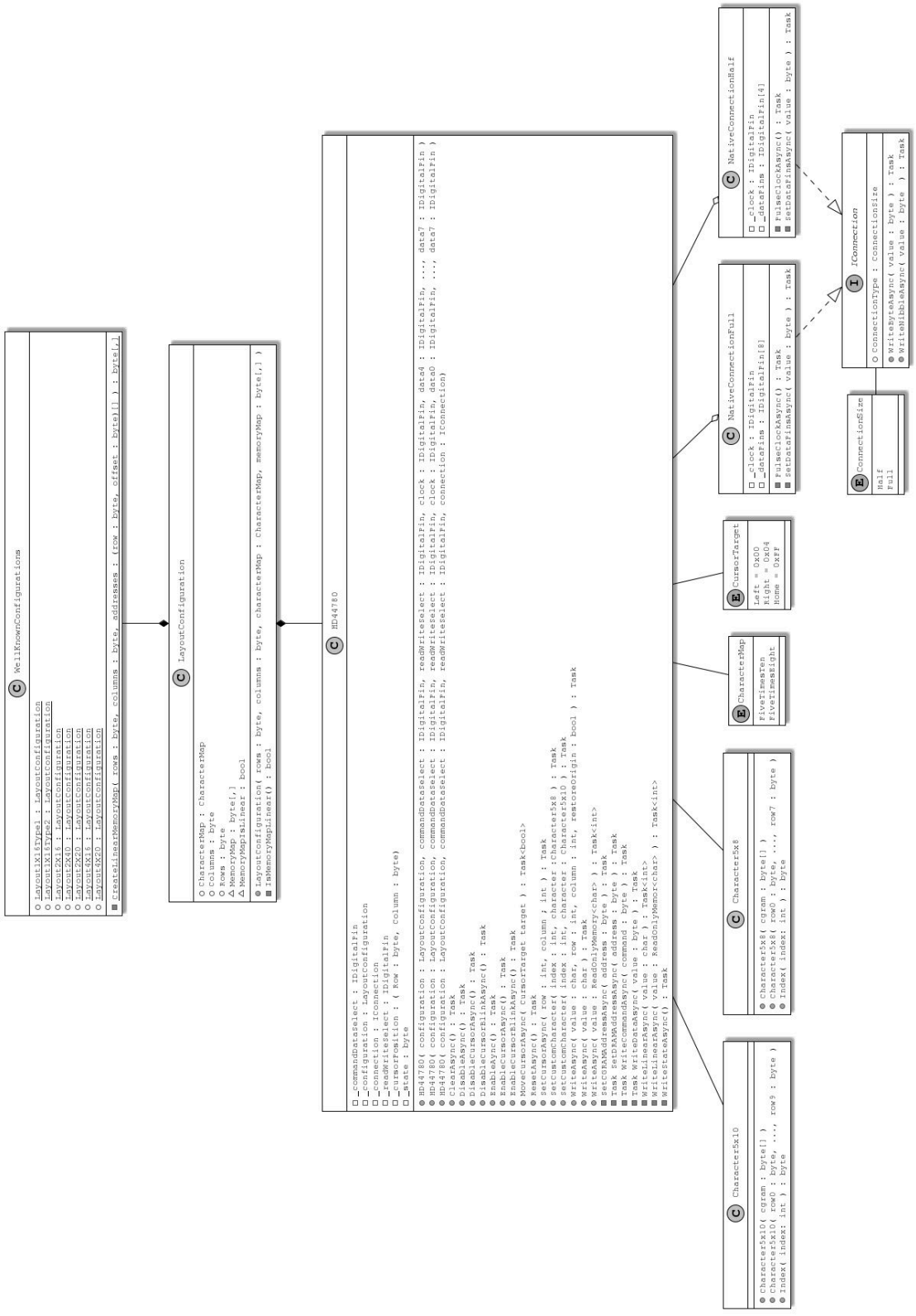

Abb. 4.4: Klassendiagramm der Bibliothek zum Ansteuern eines auf HD44780 basierenden LCD.

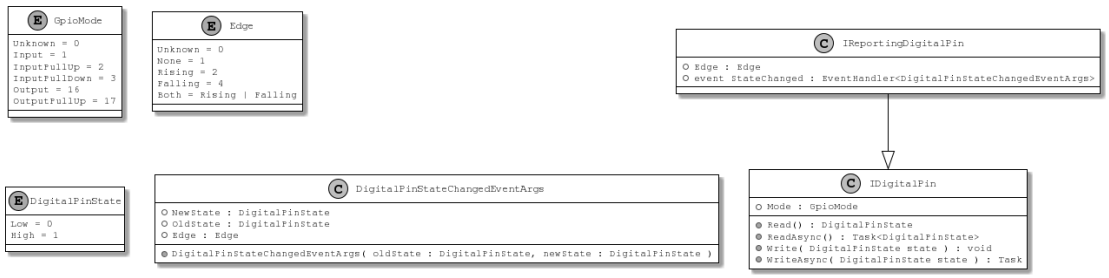

Abb. 4.5: Klassendiagramm der Bibliothek zur Abstraktion von GPIOs.

Die Grafikbibliothek, siehe Abbildung 4.7, baut auf der Display Bibliothek auf und stellt einem Verbraucher eine Schnittstelle zum Implementieren von Display Elementen bereit, `IWindow`. Diese Schnittstelle ermöglicht es einem Verbraucher auf Benutzereingaben zu reagieren und einen Framebuffer zu modifizieren. Um die Aktualisierungszeit des LCD gering zu halten werden nur Änderungen am Framebuffer an die Display Bibliothek weitergereicht.

Die Grafikbibliothek besteht aus 3 Grundelementen. Das grundlegendste Element der Grafikbibliothek ist ein Nachrichtenelement. Dieses Element erlaubt es einem Verbraucher, dem Benutzer eine Nachricht anzuzeigen währenddessen dieser auf Fertigstellung einer Aufgabe wartet. Das Nachrichtenelement ignoriert Benutzereingaben. Ein weiteres Element ist ein Dialogelement, siehe Abbildung 4.8, welches es einem Verbraucher ermöglicht einem Benutzer eine Nachricht mit verschiedenen Antwortmöglichkeiten anzuzeigen und die Antwort des Benutzers auszulesen.

Ebenso stellt die Grafikbibliothek ein Auswahlelement, siehe Abbildung 4.9, zur Verfügung. Mithilfe von diesem Element ist es einem Verbraucher möglich, einem Benutzer eine Auswahl an Elementen bereitzustellen. Diese Elemente bestehen aus einer Zuordnung von einem Label und einem Objekt, dies erlaubt die direkte Rückgabe von Objekten anstelle eines Indexes.

4.4 Netzwerkbibliothek

Zur Kommunikation zwischen Client und Server wird ein simples Netzwerkprotokoll benötigt. Existierende Netzwerkprotokolle sind meist auf einen Anwendungsfall ausgelegt und somit ungeeignet für die Wiederverwendung in einem anderen Kontext. Existierende allgemeine Protokolle wie `BEEP` sind zwar geeignet. Jedoch aufgrund fehlender Implementierungen für die verwendete Plattform würden diese die Erstellung einer Implementierung für die Plattform erfordern. Aufgrund nicht benötigter Funktionalität im Kernbefehlssatz des BEEP Protokolls ist es jedoch effizienter, ein einfaches Netzwerkprotokoll mit der benötigten Funktionalität selbst zu definieren. Das entworfene Protokoll trägt den Arbeitstitel Simple Extensible Networking Protocoll (SXNP).

Das für diese Anwendung konzipierte Netzwerkprotokoll ist ein simples Paket basiertes Anfrage-Antwort Protokoll. Das Protokoll definiert ein einfaches Paket-

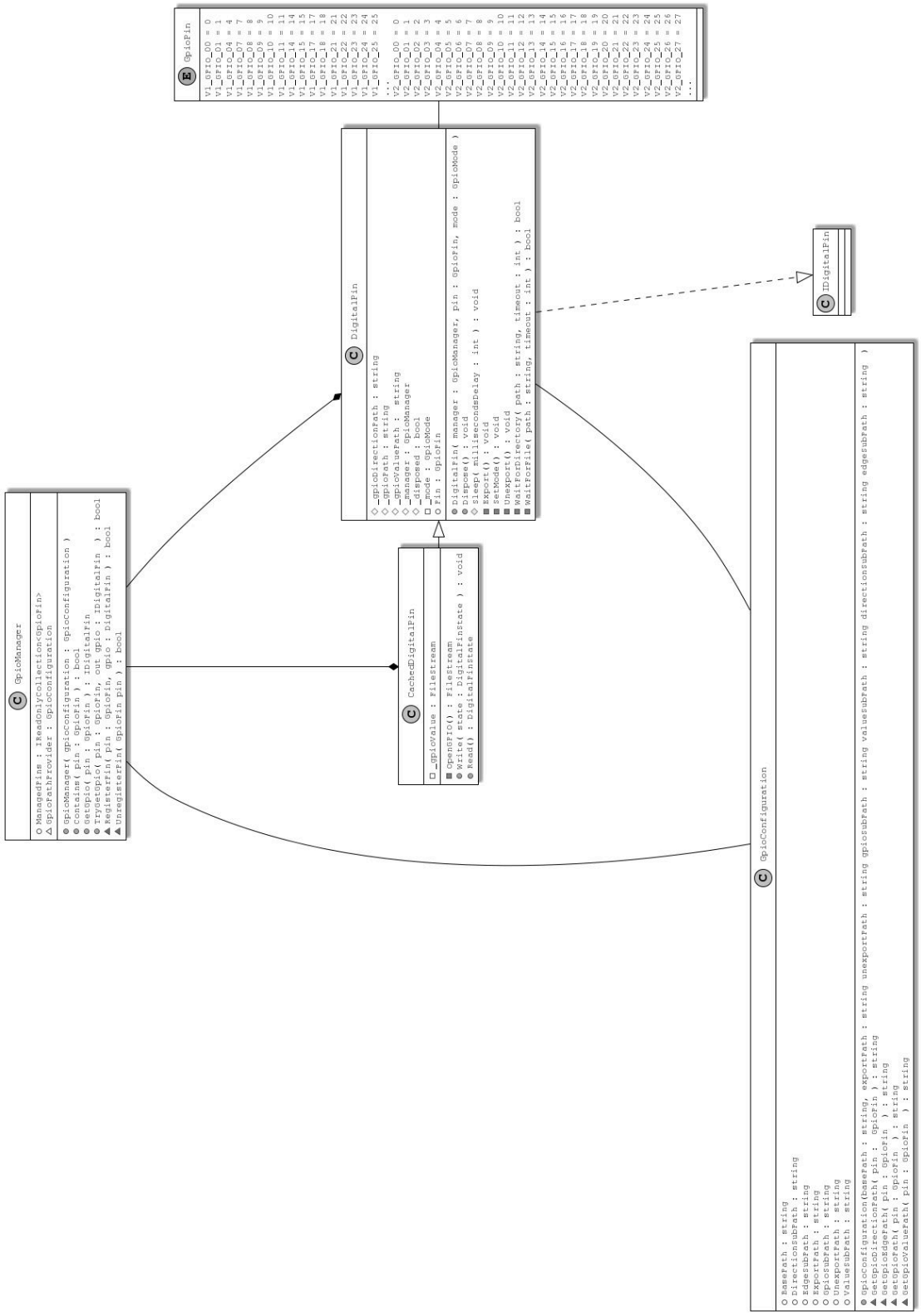

Abb. 4.6: Klassendiagramm der Bibliothek zum Ansteuern von GPIOs auf einem Raspberry Pi.

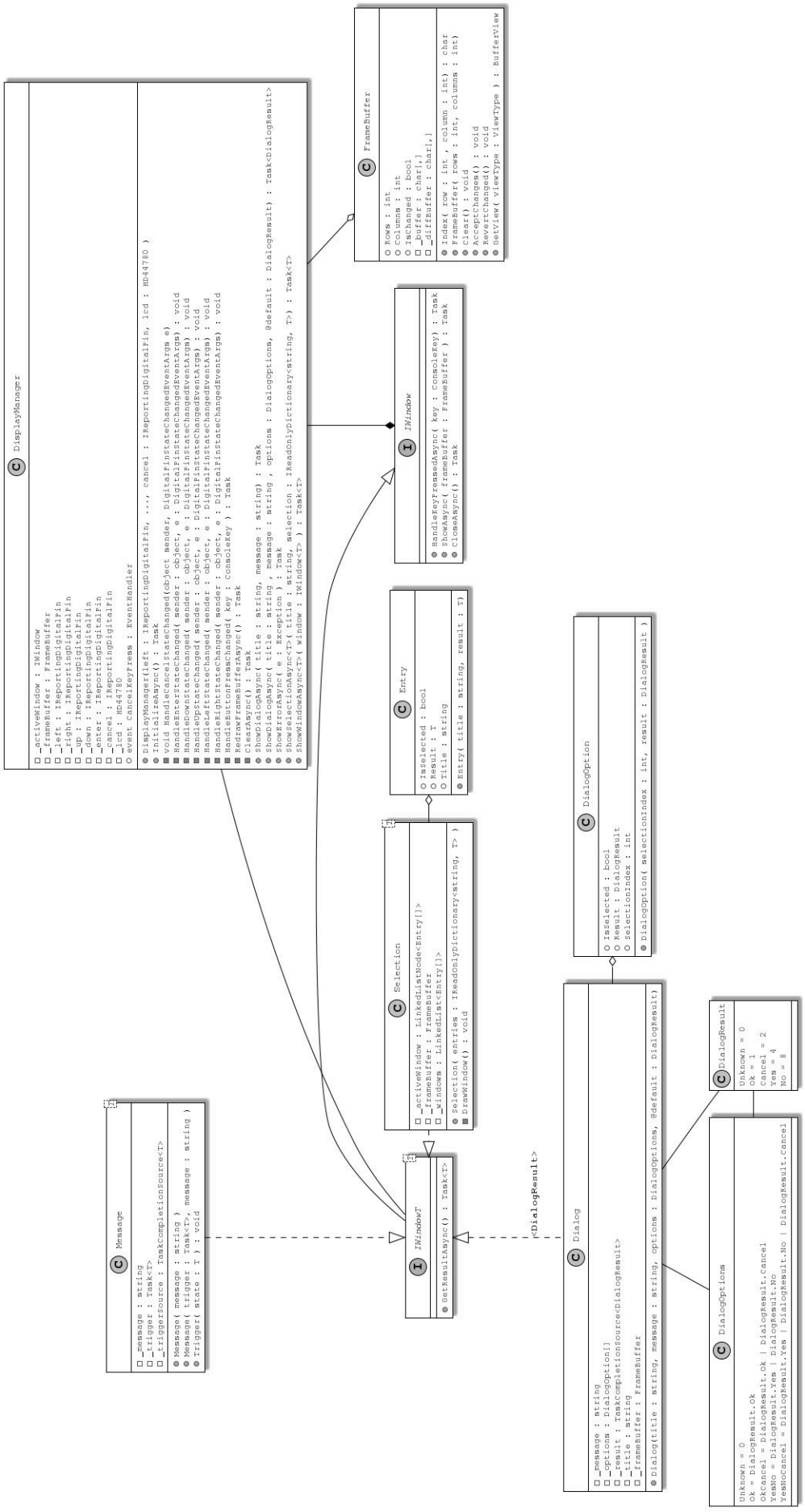

Abb. 4.7: Klassendiagramm der Grafikbibliothek.

Abb. 4.8: Beispiel Dialog der Grafikbibliothek.

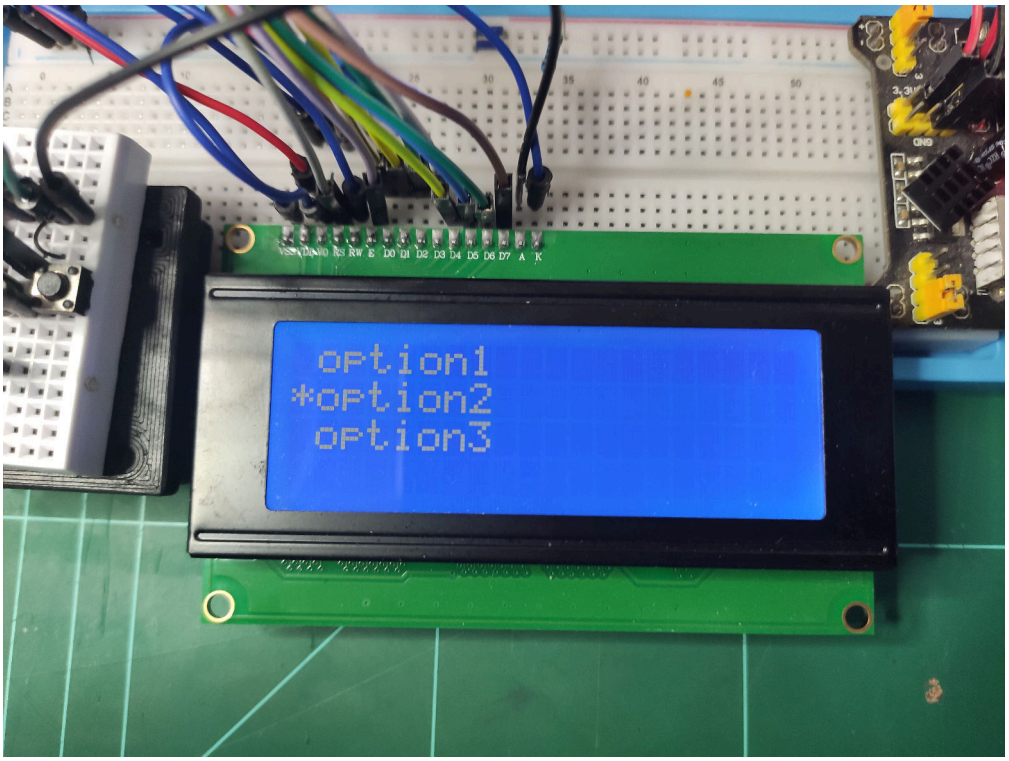

Abb. 4.9: Beispiel Auswahlelement der Grafikbibliothek.

layout, siehe Abbildung 4.10, bestehend aus einer Typendefinition, Flaggen, Paketnummer, Befehlsnummer, Nutzdatenlänge und Nutzdaten.

Offsets	**Octet**	**0**								**8**								**16**								**24**							
Octet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	**0**	Message type								RESERVED							Multipart	Message id															
4	**32**	Command/Status																Payload length															
8 …	**64** …	Payload																															

Abb. 4.10: Bytediagramm eines Netzwerkpaketes des SXNP Protokolls.

Um Nutzdaten mit einer Größe über 65KiB transferieren zu können definiert das Protokoll einen Mechanismus eine Nachricht in mehrere Pakete aufzutrennen und wieder zusammenzusetzen. Ebenso ermöglicht das Aufteilen von Nachrichten, auch kleiner als 65KiB, in mehrere kleinere Pakete einem arbeitsspeicherbegrenzten System, wie einem Mikrocontroller, diese als einzelne Nachricht zu versenden. Der generelle Verarbeitungsablauf für eingehende Pakete ist in Abbildung 4.11 dargestellt. Eingehende Pakete werden initial geprüft ob diese zu einer aufgeteilten Nachricht gehören, eine neue aufgeteilte Nachricht erstellen oder eine bestehende aufgeteilte Nachricht abschließen. Nachdem eine Nachricht eingegangen ist, wird diese anhand ihrer Typendefinition von Request, Response oder Push an ein abarbeitendes Subsystem weitergereicht. Für den Fall, dass eine Nachricht eine Antwort auf eine vorherige gesendete Anfrage ist, wird diese dieser Anfrage zugeordnet und markiert diese als abgeschlossen. Eine besondere Art der Anfrage stellt der Push dar, dieser ist eine Anfrage welche vom gegenüber nicht mit einer Antwort versehen wird. Dies ermöglicht es, Nachrichten die keine Bestätigung benötigen zu versenden.

Die Referenzimplementierung des Netzwerkprotokolls, siehe Abbildung 4.12, ist mit Unterstützung für asynchronen voll Duplex-Betrieb konzipiert. Die Netzwerkbibliothek trennt die Paketverarbeitung in mehrere Schichten auf. Auf der untersten Schicht, durch die Klasse `Connection` repräsentiert, werden Pakete von ihrer Objektrepräsentation in ihre binäre Netzwerkrepräsentation übersetzt. Diese Pakete werden, mithilfe der Klasse `MultiPartMessageManager`, in der übergeordneten Schicht aus Nachrichten erzeugt oder wieder zu Nachricht zusammengesetzt. Anfrage und Antwort Nachrichten werden durch die Klasse `ResponseManager` in der obersten Schicht, dargestellt durch die Klasse `Session`, verwaltet.

Aufgrund dessen, dass die Weboberfläche mithilfe von Webassembly innerhalb eines Browsers ausgeführt wird, kann diese aufgrund des Sicherheitsmodells von Browsern keine TCP Verbindungen aufbauen. Webassembly ist jedoch in der Lage, eine Websocket Verbindung aufzubauen. Um die Verbindung zwischen Weboberfläche und Server zu ermöglichen, stellt der Server seinen Endpunkt nicht über einen regulären TCP Server zur Verfügung, sondern über einen Webserver welcher lediglich Websocket Verbindungen bereitstellt. Die Kapselung des Protokolls in Verwendung in der konzipierten Anwendung wird in Abbildung 4.13 veranschaulicht. Damit der Konsument direkten Zugriff auf die TLS Verbindung und

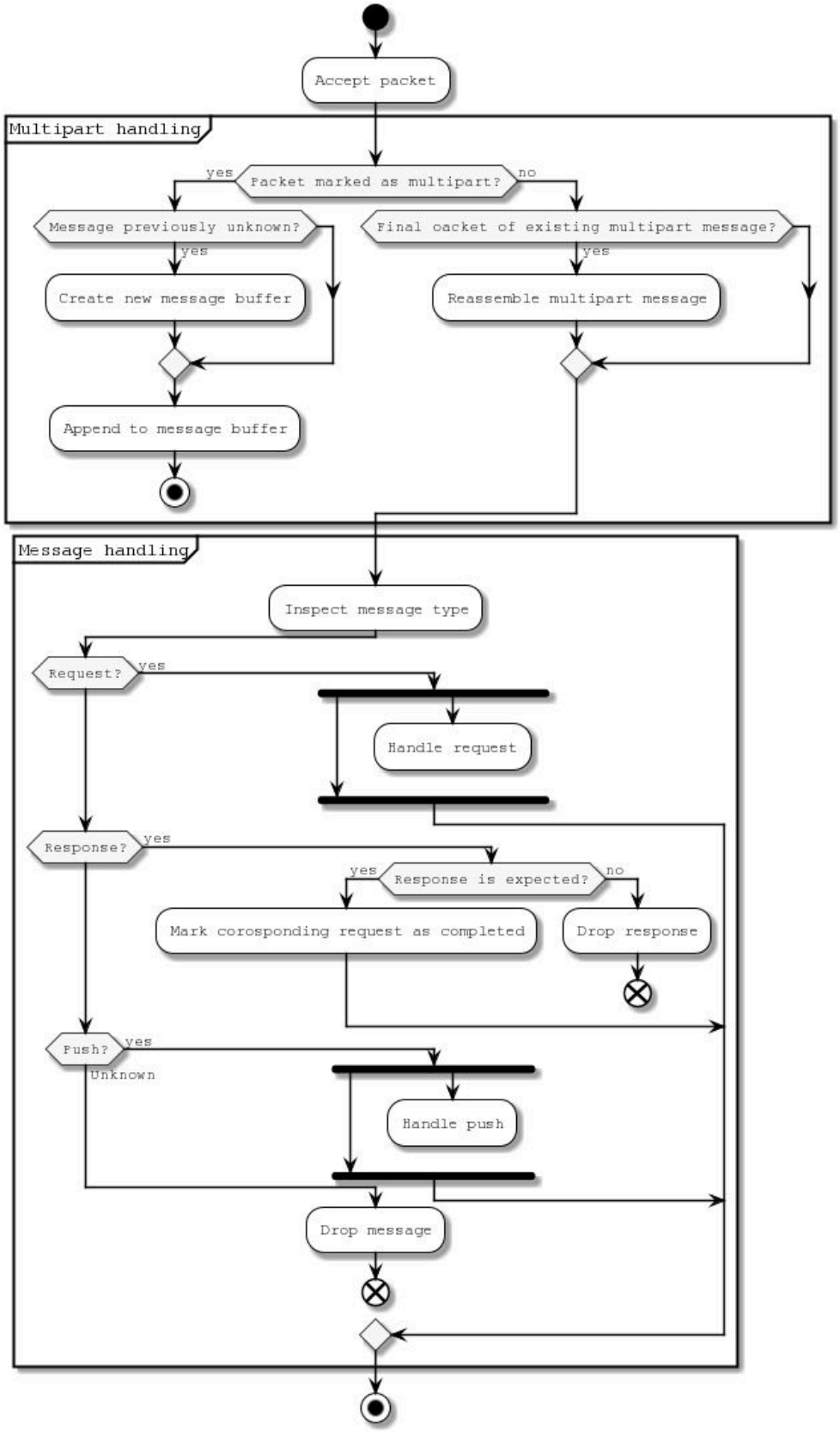

Abb. 4.11: Verarbeitungsablauf eines Netzwerkpaketes.

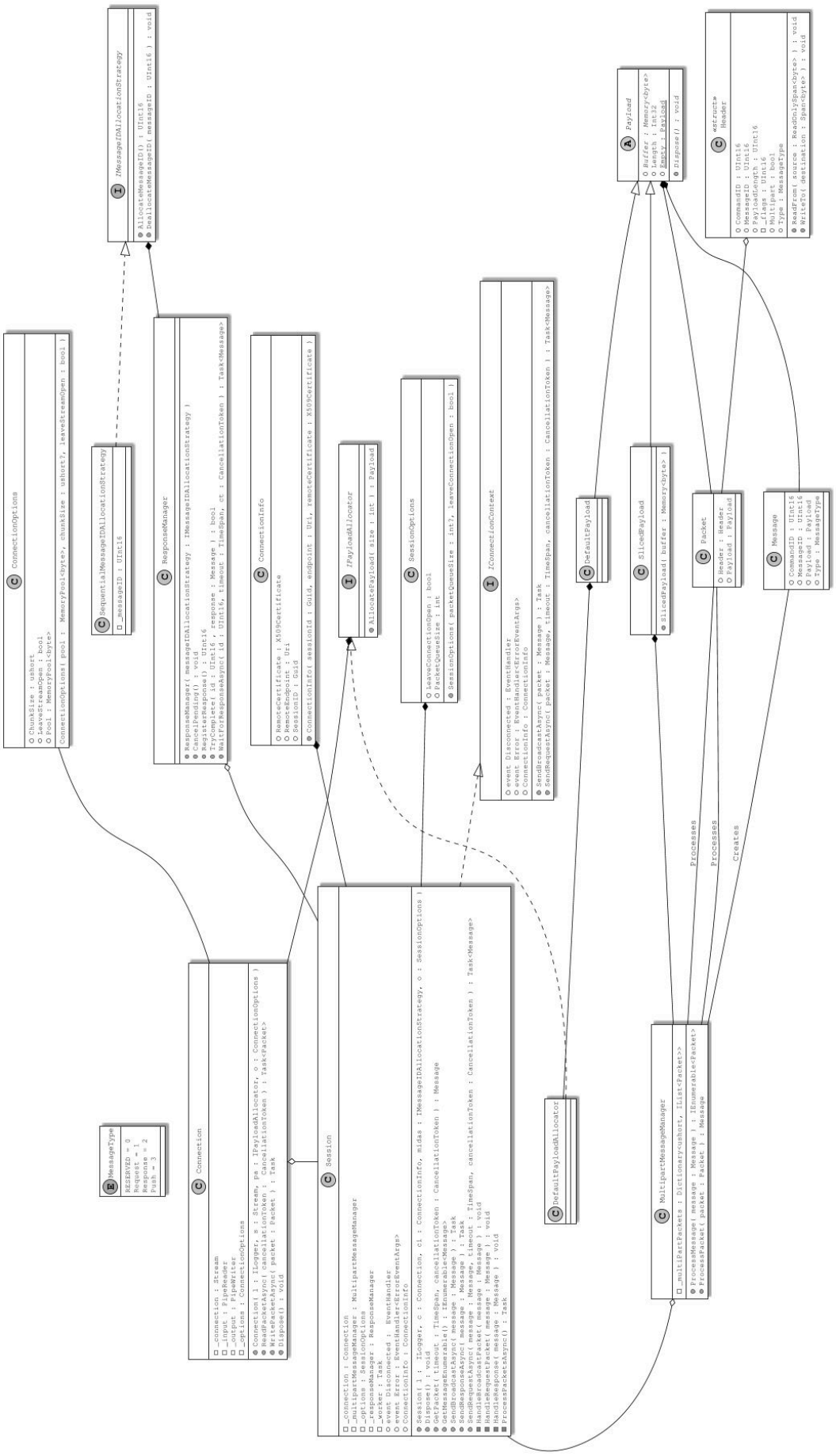

Abb. 4.12: Architektur der Netzwerkbibliothek

somit die verwendeten Zertifikate hat wird diese über einem normalen WebSocket aufgebaut und nicht unter Verwendung von WebSocket over TLS bereitgestellt.

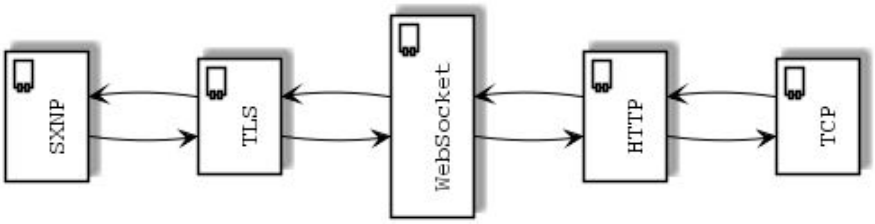

Abb. 4.13: Übersicht über die Schichtung des Protokolls

4.5 OTP Dienste

Zur Generierung von HOTP und TOTP Codes verwendet die Anwendung zwei Sets von Diensten. Der Dienst für HOTP, siehe Abbildung 4.14, generiert aus den Parametern für einen HTOP Token einen Code. Simultan verwaltet der Dienst den internen Zustand der HOTP Parameter.

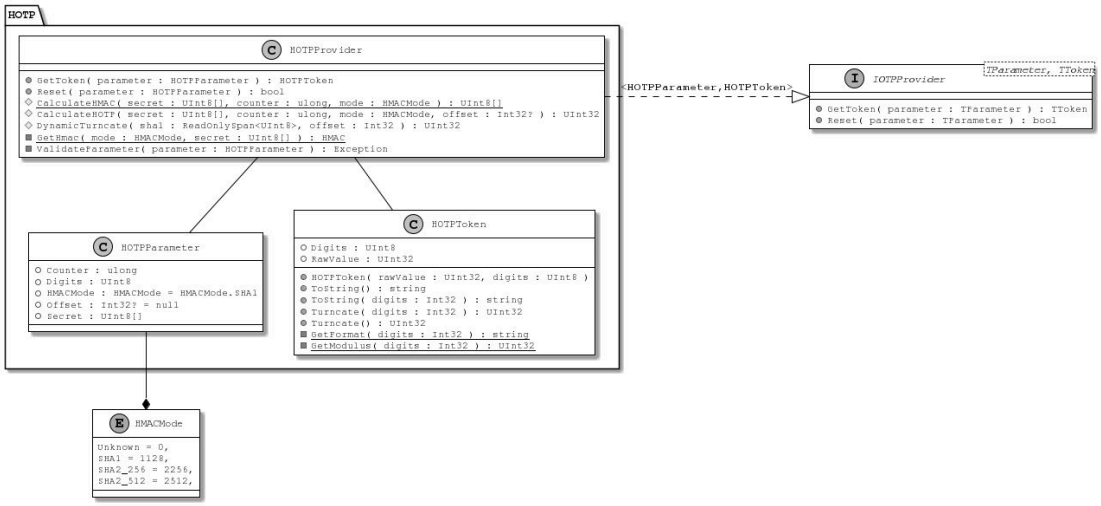

Abb. 4.14: Klassendiagramm der Bibliothek zum Generieren von HOTP.

Der Dienst, siehe Abbildung 4.15 zu generieren von TOTP Codes baut auf dem Dienst für HOTP auf. Der TOTP Dienst unterscheidet sich primär in dem Mechanismus zur Generierung des Counters zu Codegenerierung. Zur Persistierung der Parameter und Zustände werden Token, sowohl HOTP und TOTP zusammen, in einem indexierten ZIP-Archive abgespeichert. Persistiert werden hierbei lediglich die Parameter und Kennung eines Token. Die aus diesem Token generierten Codes werden nicht persistiert.

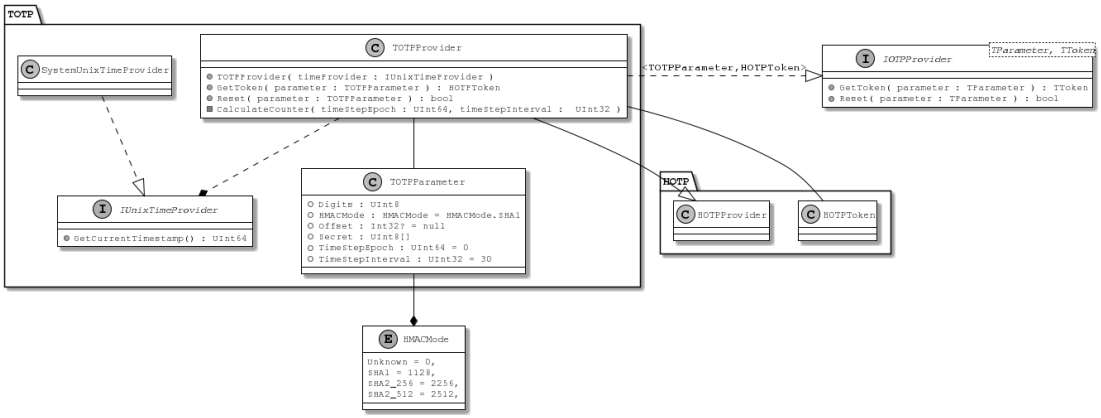

Abb. 4.15: Klassendiagramm der Bibliothek zum Generieren von TOTP.

Das Layout, zu sehen in Abbildung 4.17, der indexierten ZIP Datei ist ein simples anhand der UUID eines Eintrages indexiertes Verzeichnis. Zur einfachen Referenzierung der vorhandenen Einträge werden diese in einem zentralen Index anhand ihrer UUID aufgelistet. Zum Zugriff auf diese Einträge werden diese über ein simples Create, Read, Update and Delete (CRUD) Repository verwaltet, siehe Abbildung 4.16. Dieses stellt einem Konsumenten eine typenunabhängige Auflistung aller Einträge, dargestellt durch die Klasse `OTPEntry`, ohne Parameter zu Verfügung. Ein Eintrag besteht aus einer eindeutigen Kennung in Form einer UUID, einem Namen und einer Typbeschreibung der Parameter. Erweitert werden diese Einträge um Einträge, dargestellt durch die Klasse `OTPEntry<T>`, welche um die Algorithmenparameter, wie das geteilte Geheimnis und Zeichenanzahl, angereichert sind. Um die geteilten Geheimnisse nur notwendig lang im Arbeitsspeicher zu halten werden diese Einträge angereichert um Parameter nur nach Bedarf geladen und anschließend über Dotnet eigene Mechanismen wieder aus dem Arbeitsspeicher entfernt; die Parameter werden dereferenziert und vom Garbage Collector eingesammelt. Um den nicht Determinismus des Garbage Collector zu umgehen, wird dieser umgehend nach Dereferenzierung manuell ausgeführt. Anstelle der Verwendung von managed Memory könnte unmanaged Memory mit einer manuellen Säuberung zur Speicherung des geteilten Geheimnisses verwendet werden; dies kann jedoch zu Referenzierungsfehlern führen und wird nicht von dem Standard IO Bibliotheken unterstützt.

4.6 Benutzerschnittstelle

Die allgemeine Benutzerschnittstelle ist der primäre Einstiegspunkt für die konzipierte Anwendung. Die konzipierte Anwendung, siehe Abbildung 4.18, verwendet die Dependency-Injection Bibliothek von Microsoft zum automatischen auflösen der Dependencies der Initialisierten Klassen. Ebenso verwaltet die Dependency-Injection Bibliothek die Lebenszeit der einzelnen Komponenten der Anwendung sowie der Anwendung selbst. Des weiteren wird durch die Dependency-Injection

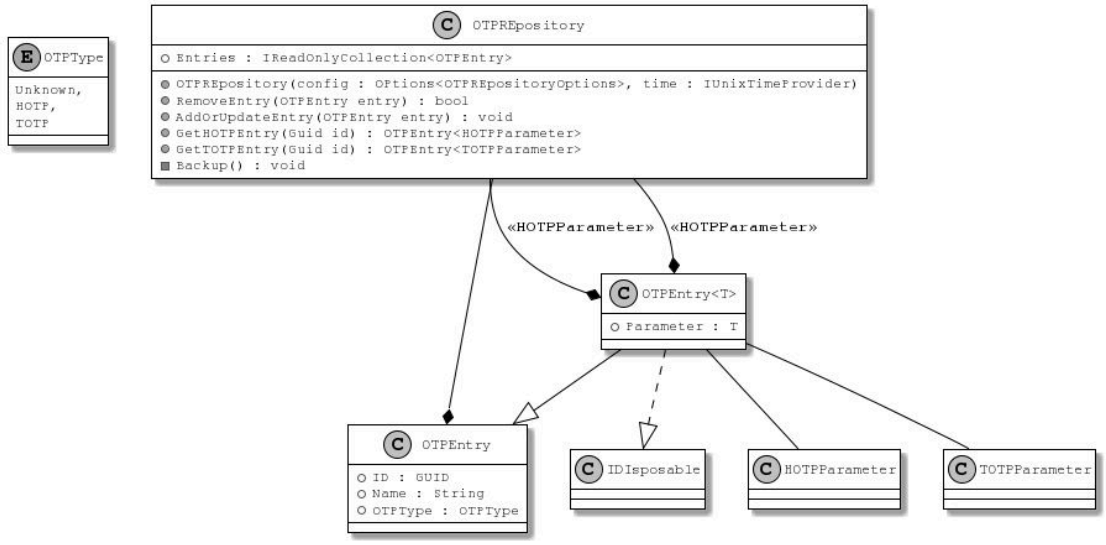

Abb. 4.16: Klassendiagramm der CRUD Bibliothek zum Verwalten von HOTP/-TOTP Einträgen.

```
/
\-entries/
  |-INDEX
  |-b514ec4573a3431fb314294645123c41
  \-db74011fc2a94c7c8c45e7060a847591
```

Abb. 4.17: Layout innerhalb der ZIP Datei genutzt zur Persistierung von Daten mit zwei Beispieleinträgen.

Bibliothek die Konfiguration der Anwendung eingelesen, geparst und in die Instanzen injiziert. Während dem Startvorgang der allgemeinen Benutzerschnittstelle wird der Benutzer aufgefordert, die lokale oder entfernte Schnittstelle zu starten. Die gewählte Schnittstelle wird von der allgemeinen Benutzerschnittstelle in ihren Ladevorgang übernommen und gestartet.

Die Konfiguration der Anwendung umfasst plattformspezifische Einstellungen betreffend der Hardwareschnittstelle des LCDs und dem Speicherort der Token Datenbank.

4.7 Zeitsynchronisation

Um TLS Zertifikate zu verifizieren und TOTP Token erstellen zu können benötigt die Anwendung die aktuelle Zeit in Form eines 64Bit Unix Zeitstempels. Ein Unix Zeitstempel ist ein 32Bit oder 64Bit Wert der Sekunden seit der UNIX Epoche 1970-01-01T00:00:00 darstellt. Auf Desktopsystemen wird meist die Zeit mittels eine auf dem Motherboard verbauten Echtzeituhr zwischen Neustarts per-

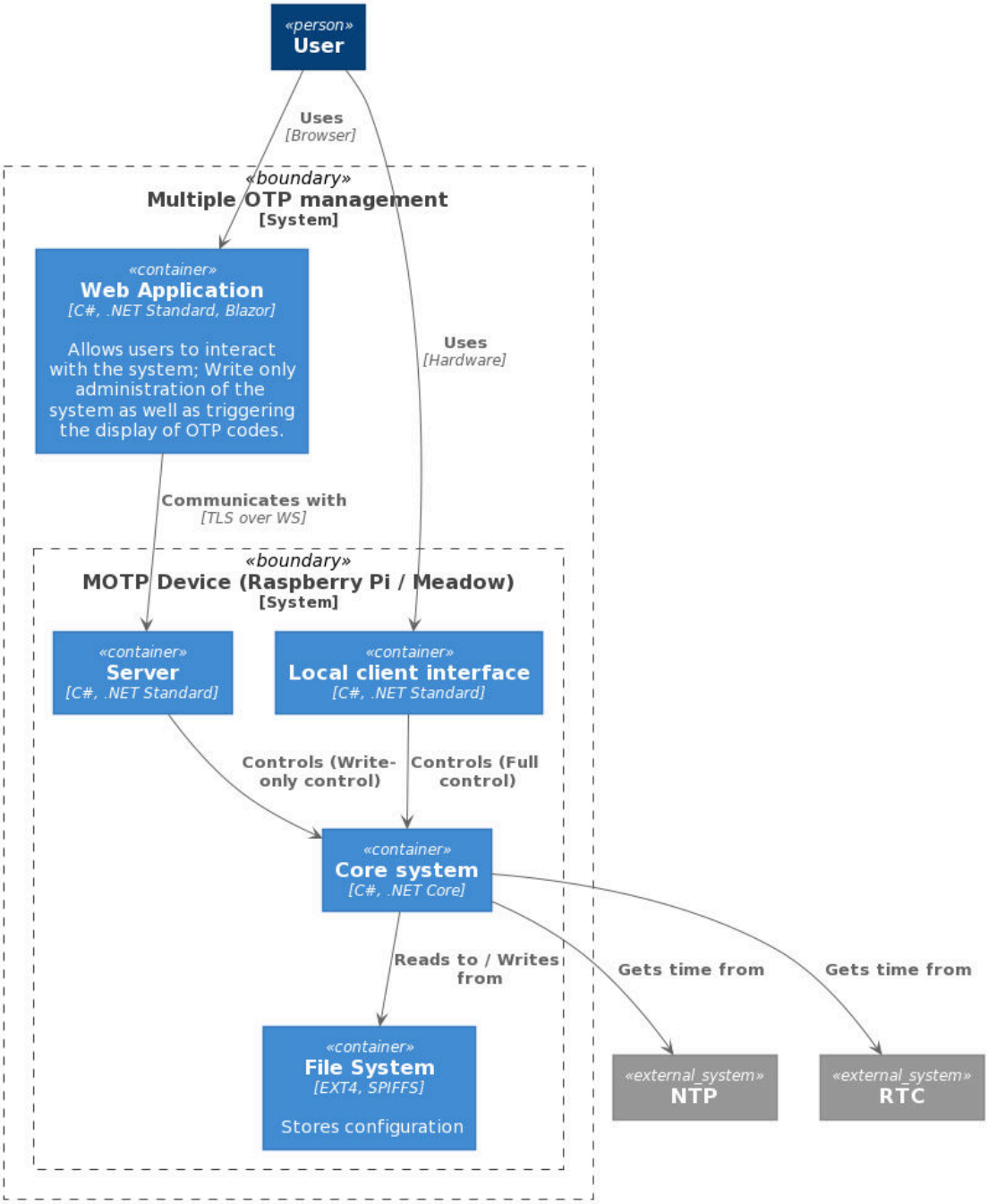

Abb. 4.18: C4-Diagramm Darstellung der Architektur der konzipierten Anwendung.

sistiert und mittels NTP aktualisiert. Weder Meadow noch Raspberry Pi werden standardmäßig mit einer Echtzeituhr ausgestattet. Um auf diesen Systemen dennoch einen aktuellen Zeitstempel zu erhalten, müssen diese mit einer Echtzeituhr mit Batterieunterstützung ausgestattet werden. Auf der Meadow Plattform muss die Anwendung die Echtzeituhr direkt über I2C ansprechen und verwalten. Währenddessen auf der Raspberry Pi Plattform die Echtzeituhr, siehe Abbildung 4.19, durch das unterliegende Linuxsystem verwaltet wird. Das Linuxsystem stellt den Zeitstempel von der Echtzeituhr Anwendungen zur Verfügung und aktualisiert die Echtzeituhr mittels NTP wenn eine Netzwerkverbindung besteht.

Die für den Prototypen verwendete Echtzeituhr, siehe Abbildung 4.19, basiert auf dem DS3231 Chip. Die verwendete Echtzeituhr verfügt über eine Batterieunterstützung um die Zeit im DS3231 auch bei externem Spannungsverlust aktuell zu halten. Der DS3231 Chip bietet Schaltsekunden Korrektur bis zu dem Jahr 2100 und hat unter Normalbedingungen eine Uhrzeitverschiebung von +- 2 Minuten pro Jahr [11].

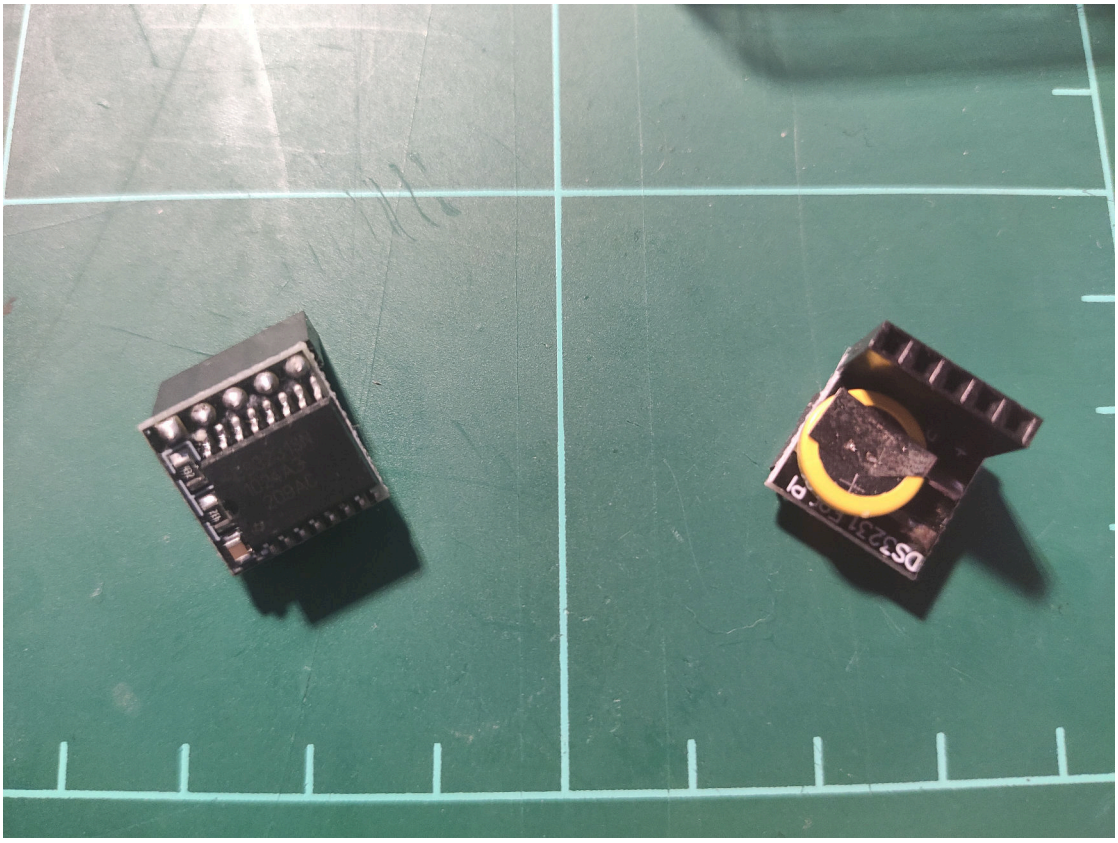

Abb. 4.19: Über I2C angesteuerte Echtzeituhr basierend auf dem DS3231.

4.8 Testen

Zum Testen der Softwarekomponenten der konzipierten Anwendung wurden für die verschiedenen Komponenten Unit-Tests verwendet. Die Unit-Tests wurden mithilfe der NUnit Bibliothek realisiert. Diese Bibliothek wiederum ist inspiriert von der bekannten Java Bibliothek JUnit. Die Bibliothek stellt ein Testframework basierend auf attributisierten Klassen und Methoden bereit welche einen Testfall oder einen Testablauf darstellen können. Um das Testen der Komponenten zu ermöglichen, besonders für diejenigen Komponenten, welche mit der Hardware interagieren, wurde die konzipierte Anwendung mit besonderem Augenmerk auf die Designprinzipien von Dependency Inversion und Dependency Injection entwickelt.

Im Beispiel Testfall, dargestellt in Abbildung 4.20, wird mittels NUnit3 die Codegenerierung von TOTP Token zu verschiedenen Zeitpunkten getestet. Um die Tests unabhängig von der Systemzeit durchzuführen werden mittels Dependency Injection eine alternative Implementierung des Zeitstempel Dienstes bereitgestellt. Diese Mock-Implementierung stellt sequentiell die für die Testfälle relevanten Zeitstempel zur Verfügung. Der Testfall selbst erstellt neue TOTP Parameter anhand der Testfalldaten. Zum Validieren werden die erwarteten Ergebnisse mit den erzeugten Ergebnissen verglichen. Die Testfalldaten werden mittels Reflection anhand des Attributes TestCaseSource zugeordnet.

Manuelle Testfälle

Aufgrund der Interaktion mit physikalischen Objekten in Form der Hardware sind nicht alle Testfälle automatisiert. Das automatische Testen von Hardware erfordert einen aufwändigen Aufbau zum Einlesen des Zustandes der Hardware nach einem Testfall. Daher wurden solche Testfälle lediglich durch manuelle Testaufbauten verifiziert.

4.8.1 Analyse der Anforderungen

Nachfolgend werden die zuvor festgelegten Anforderungen getestet. Die Anforderungen Req. 1, Req. 6 und Req. 8 sind erfüllt und werden in den Kapiteln 4.6, 5, 5.0.1 und 5.0.2 beschrieben. Die Anforderung Req. 2 ist durch die Verwendung einer Hardware Echtzeituhr, beschrieben in Kapitel 4.7, erfüllt. Die Anforderung Req. 3 und Req. 4 ist erfüllt und wird in Kapitel 4.5 beschrieben. Die Anforderung Req. 9 wird in Kapitel 4.8 erfüllt. In Kapitel 6 wird die Anforderung Req. 7 analysiert und erfüllt. Anforderung Req. 5 ist partiell implementiert, Backups auf den lokalen Datenspeicher sind in Kapitel 4.5 implementiert; Backups über die Weboberfläche sind implementiert, verfügen jedoch noch über keine Benutzeroberfläche zur Interaktion.

```
class TOTPProviderTests
{
  public static IEnumerable<(string hexSecret, byte digits,
   (long, string)[] expected)> TokenSequenceTestCases()
  {
    yield return ("31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30",
     8, new[]
    {
        (59, "94287082"), (1111111109, "07081804"),
        (1111111111, "14050471"), (1234567890,"89005924"),
        (2000000000, "69279037"), (20000000000, "65353130")
      });
  }

  [Test]
  [TestCaseSource(nameof(TokenSequenceTestCases))]
  public void TestTokenSequenceGenneration((string hexSecret, byte digits,
   (long, string)[] expected) testVector)
  {
    (string hexSecret, byte digits, (long Time, string Token)[] expected)
    = testVector;
    var param = new TOTPParameter()
    {
      Secret = Hex.FromString(hexSecret),
      Digits = digits,
      Offset = null
    };
    var provider = new TOTPProvider(
      new SequenceUnixTimeProvider(
        expected.Select(s => s.Time)));
    Assert.Multiple(() =>
    {
      for (int i = 0; i < expected.Length; i++)
      {
        Assert
        .AreEqual(expected[i].Token, provider.GetToken(param)
        .ToString());
   }
   });
  }
}
```

Abb. 4.20: Testfall mit NUnit3 zur Verifikation der generierten OTP Code Sequenzen mittels TOTP.

5

Bedienung

Die konzipierte Anwendung, vorübergehend bezeichnet als Multiple OTP Management Interface (MOTP), kann von dem Benutzer über die lokale oder entfernte Schnittstelle bedient werden. Zum Systemstart wählt der Benutzer die zu startende Schnittstelle aus, siehe Abbildung 5.1.

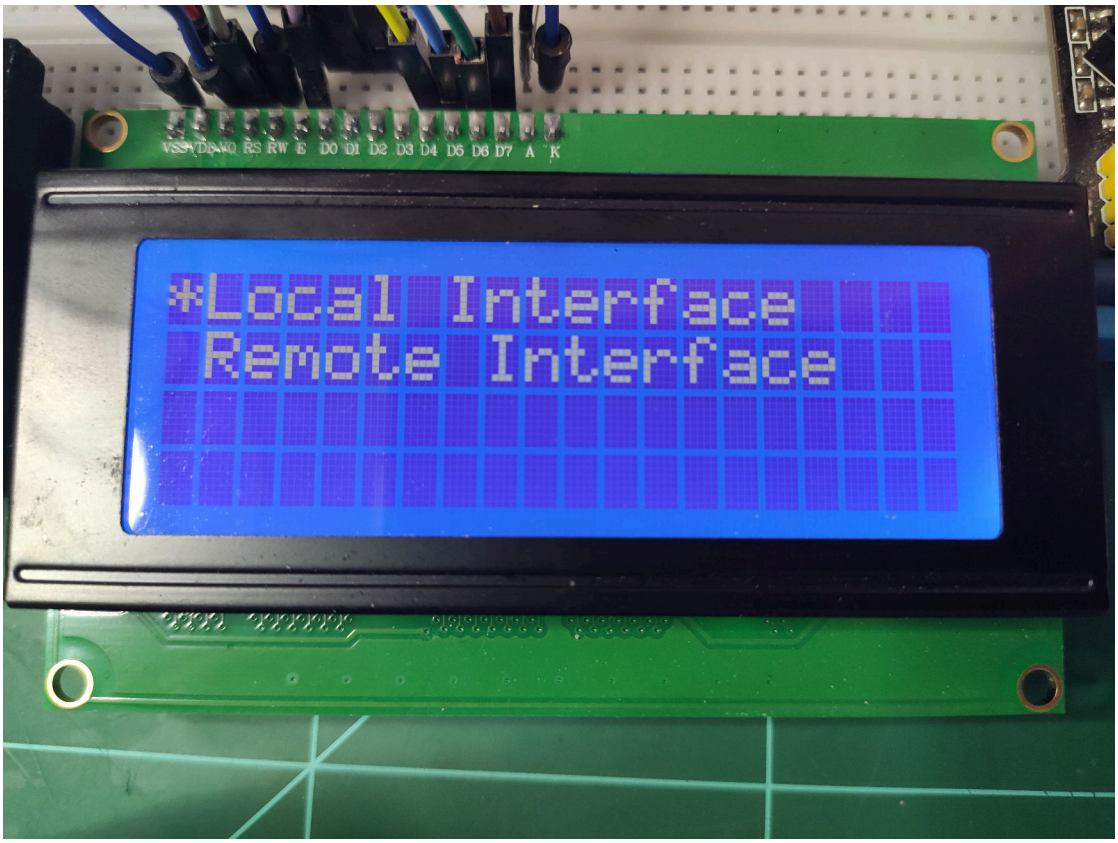

Abb. 5.1: Auswahl der verwendeten Schnittstelle zum Systemstart.

5.0.1 Lokale Schnittstelle

In der lokalen Schnittstelle wird dem Benutzer eine Auswahl an Aktionen, siehe Abbildung 5.2, zur Verfügung gestellt. Nach Durchführung jeder Aktionskette, mit Ausnahme von der Beendigung der Anwendung, wird dem Benutzer diese Auswahl erneut bereitgestellt.

Abb. 5.2: Auswahl der durchzuführenden Aktion.

Nach Auswahl einer auf einem bestehenden Token durchzuführenden Aktion wird dem Benutzer eine Auswahl über die bestehenden Token bereitgestellt, siehe Abbildung 5.3.

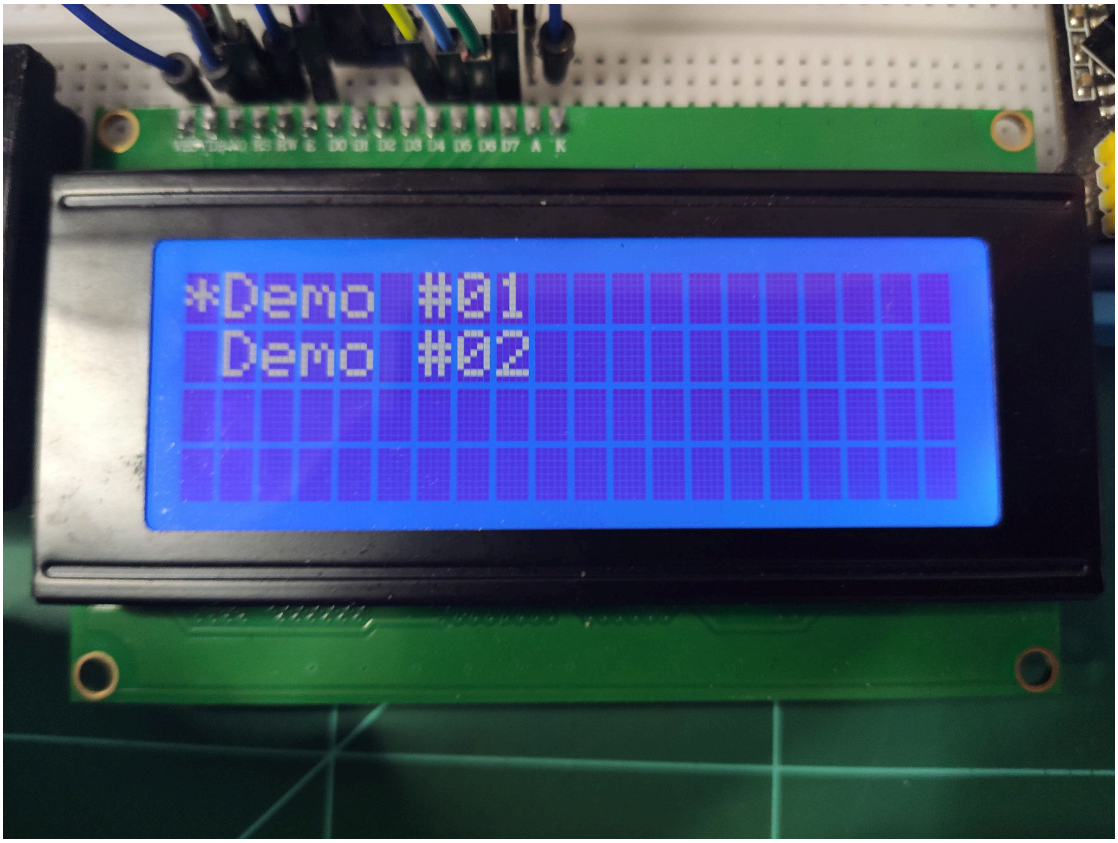

Abb. 5.3: Auswahl des Token zum Durchführen der Aktion.

Bei Auswahl der Aktion Show OTP wird dem Benutzer nach Auswahl des Token das zu dem Token zugeordnete One Time Password angezeigt, siehe Abbildung 5.4.

Abb. 5.4: Anzeigen des One Time Password des gewählten Tokens.

Bei Auswahl der Aktion Edit Token wird dem Benutzer nach Auswahl des Token eine Unterauswahl an Aktionen zum Editieren des Token angezeigt, siehe Abbildung 5.5. In der Unterauswahl zum Editieren des Token kann der Benutzer den Token Löschen, Zurücksetzen oder dessen Parameter editieren.

Abb. 5.5: Unterauswahl der durchzuführenden Aktion am Token.

5.0.2 Entfernte Schnittstelle

Wählt der Benutzer beim Systemstart aus, die entfernte Schnittstelle zu starten, werden auf der lokalen Schnittstelle Systeminformationen angezeigt, siehe Abbildung 5.6. Zu den Systeminformationen gehört die IP-Adresse zu welcher sich die entfernte Schnittstelle verbinden kann.

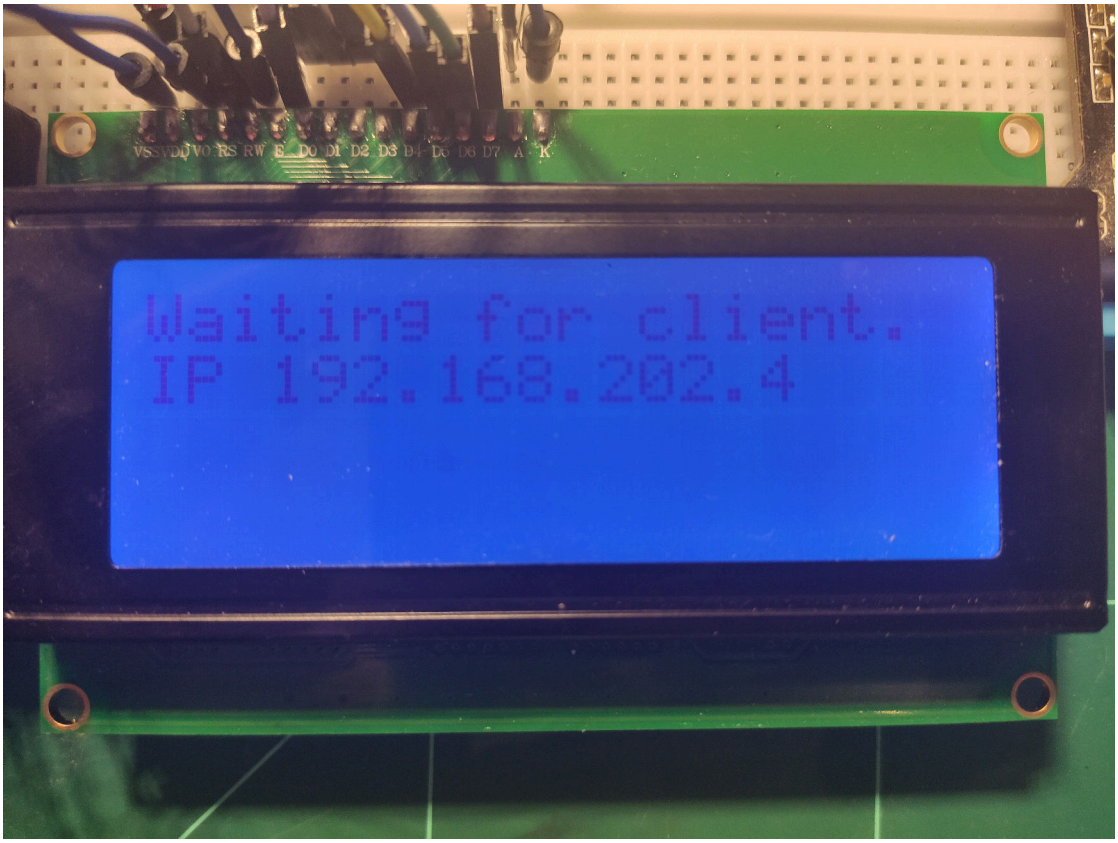

Abb. 5.6: Anzeige des Systemstatus auf der lokalen Schnittstelle während dem Warten auf einen Verbindungsaufbau.

Von der entfernten Schnittstelle kann der Benutzer die Verbindung zu dem System aufbauen. Während dem Verbindungsaufbau wird der Fingerabdruck, siehe Abbildung 5.7, der Schnittstelle für den späteren Abgleich angezeigt.

Der im Prototypen verwendete Fingerabdruck ist der mittels MD5 reduzierte Hash des SHA256 Fingerabdruckes des Zertifikats. Zwar ist MD5 nicht länger sicher, jedoch wird das Zertifikat nur für einen Verbindungsversuch verwendet und anschließend neu generiert. Die Kurzlebigkeit des Zertifikats reduziert weiterhin das mögliche Angriffsfenster. Sichere Hashfunktionen wie SHA-256 lassen sich nicht vollständig auf dem LCD anzeigen und sind somit ungeeignet.

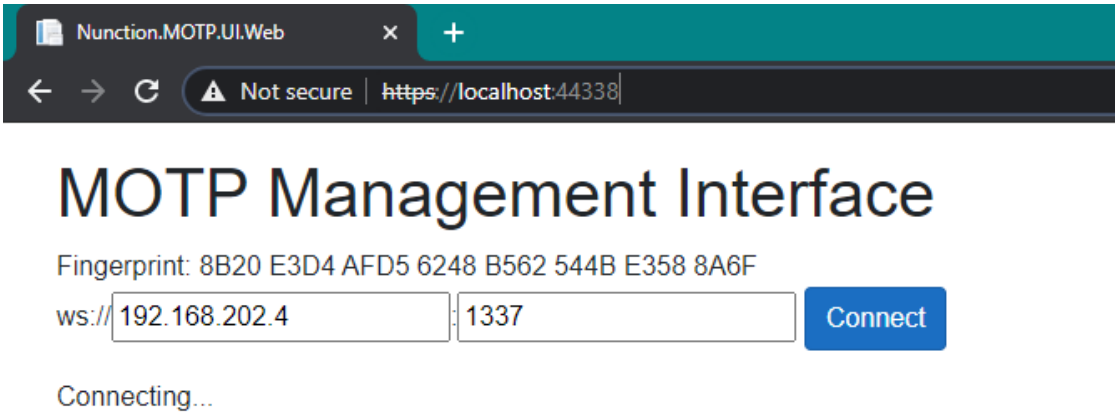

Abb. 5.7: Verbinden der entfernten Schnittstelle zum System.

Nachdem eine entfernte Schnittstelle die Verbindung zu dem System initialisiert hat und die Verbindung gesichert wurde, wird der Benutzer aufgefordert, die Identität der Verbindung zu bestätigen, siehe Abbildung 5.8. Bestätigt der Benutzer die Identität anhand des Fingerabdruckes, siehe Abbildungen 5.7 und 5.8, der entfernten Schnittstelle wird die Verbindung aufgebaut. Lehnt der Benutzer die Identität der Schnittstelle ab, wird die Verbindung zu dieser terminiert.

Abb. 5.8: Bestätigung der Verbindung der entfernten Schnittstelle zum System auf der lokalen Schnittstelle.

Nachdem der Verbindungsaufbau zum System abgeschlossen ist wird dem Benutzer eine Auflistung der vorhandenen Token aufgelistet, siehe Abbildung 5.9. In der Auflistung werden dem Benutzer Trigger für durchführbare Aktionen für die verschiedenen Token bereitgestellt. Triggert ein Benutzer das Anzeigen eines Tokens wird der Code in der Hardware generiert und auf dieser angezeigt, siehe Abbildung 5.4, jedoch nicht zur Webbasierten entfernten Schnittstelle übertragen.

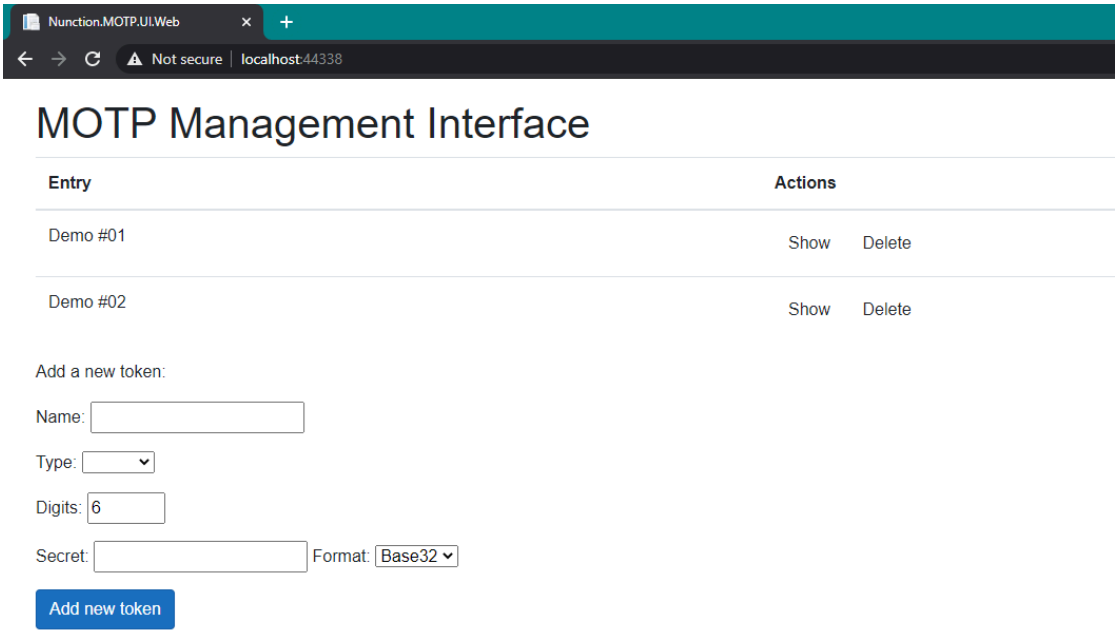

Abb. 5.9: Auflistung der gespeicherten Token in der entfernten Schnittstelle.

Unterhalb der Auflistung der bestehenden Token wird dem Benutzer ein Formular zum Hinzufügen neuer Token bereitgestellt, siehe Abbildung 5.11. In dieses Formular werden das geteilte Geheimnis, ein identifizierender Name und Algorithmenparameter eingetragen. Diese Parameter werden von einer gesicherten elektronischen Ressource bereitgestellt, siehe Abbildung 5.10. Nach Hinzufügen des Token, siehe Abbildung 5.12, wird ein Code generiert und als Bestätigung zur gesicherten elektronischen Ressource übertragen. Sofern dieser mit dem erwarteten übereinstimmt wird Two-Factor Authentication aktiviert, siehe Abbildung 5.13.

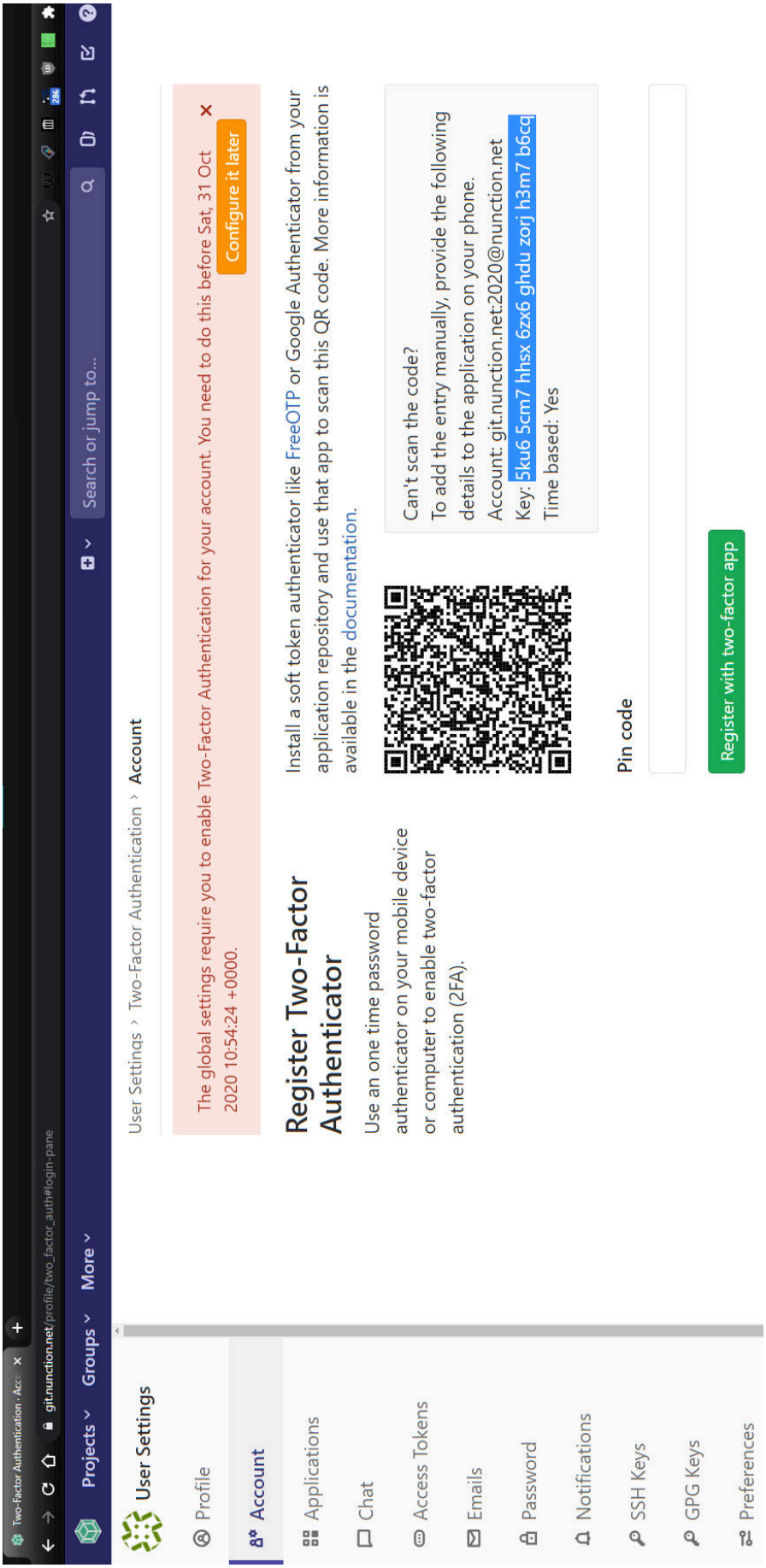

Abb. 5.10: Sicherheitseinstellung eines Accounts in GitLab zum Hinzufügen eines Two-Factor Authenticator. In Blau markiert das geteilte Geheimnis zum Hinzufügen zum Security-Token.

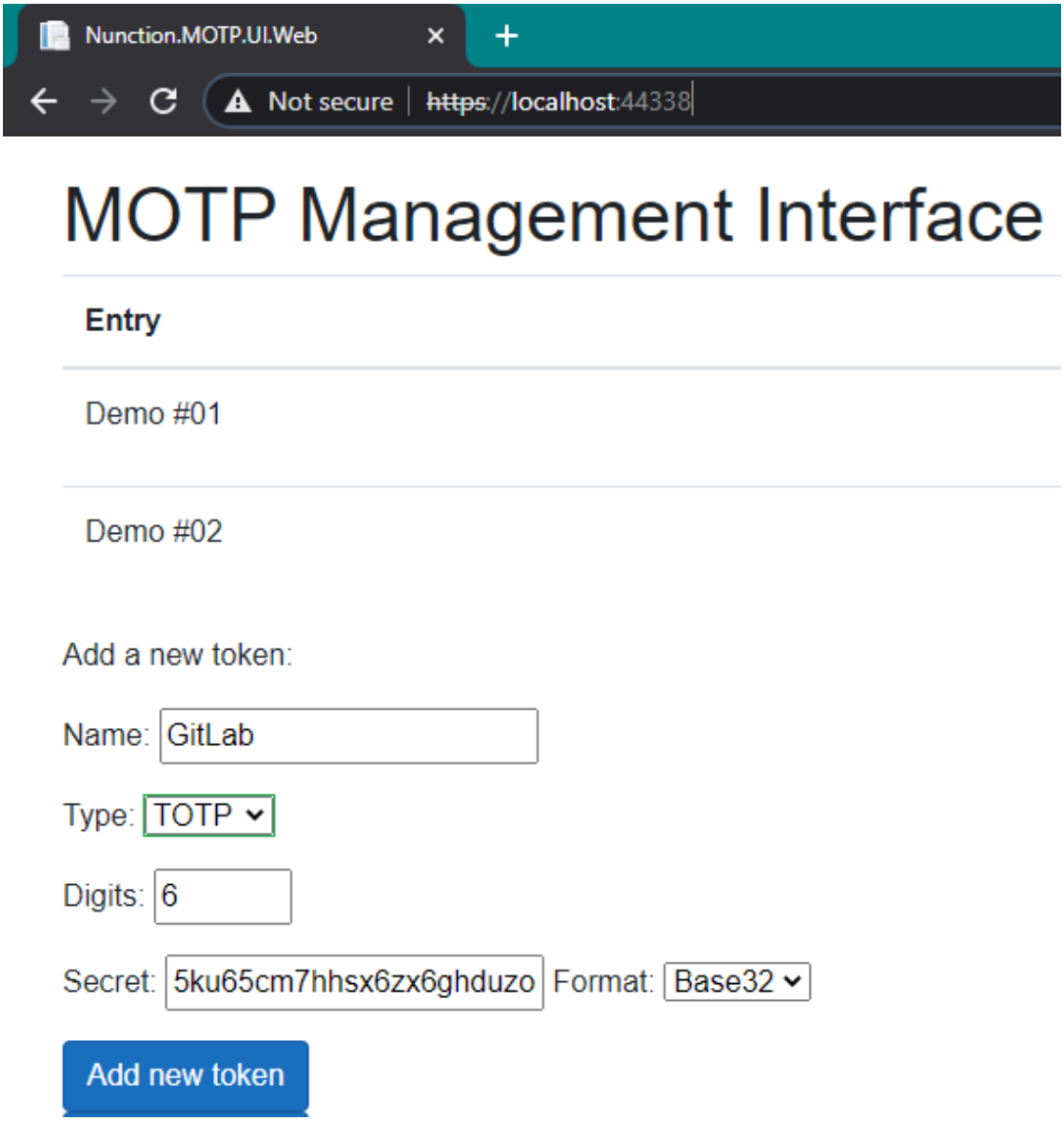

Abb. 5.11: Hinzufügen eines neuen Token zum System von der entfernten Schnittstelle.

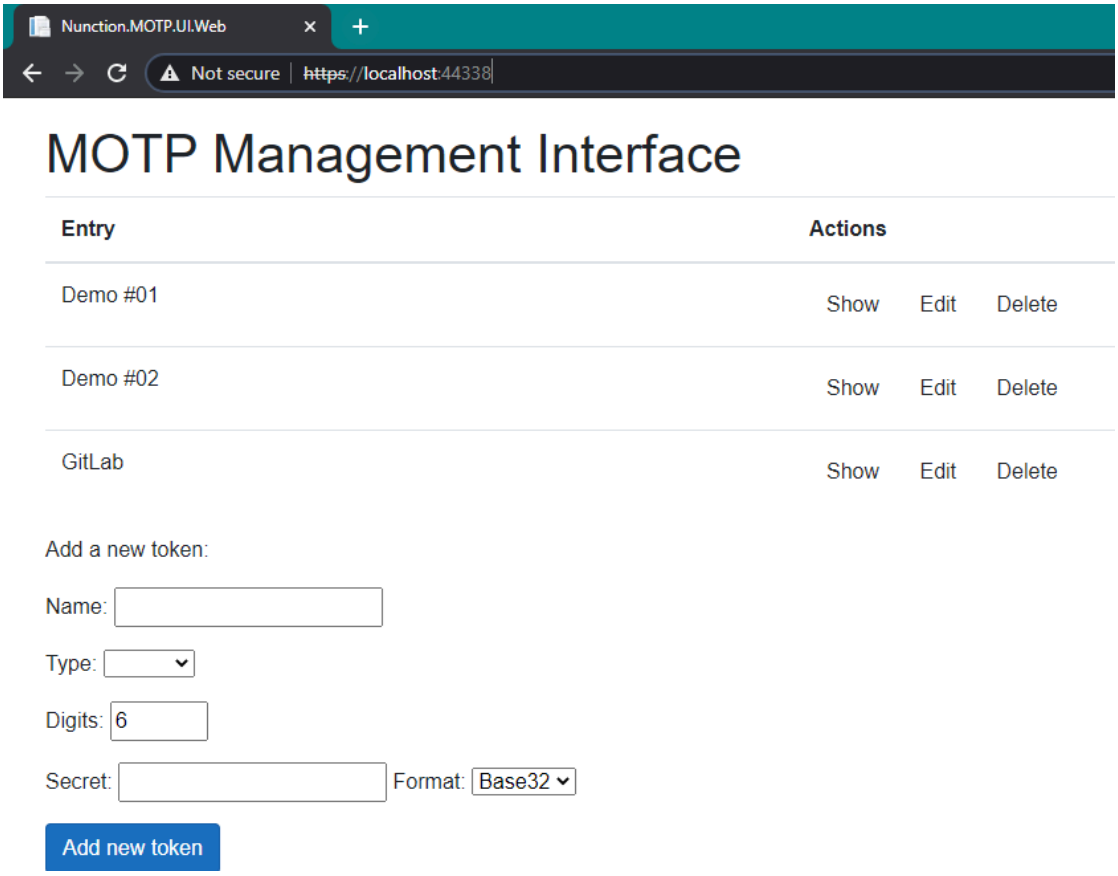

Abb. 5.12: Neu hinzugefügter Token im System.

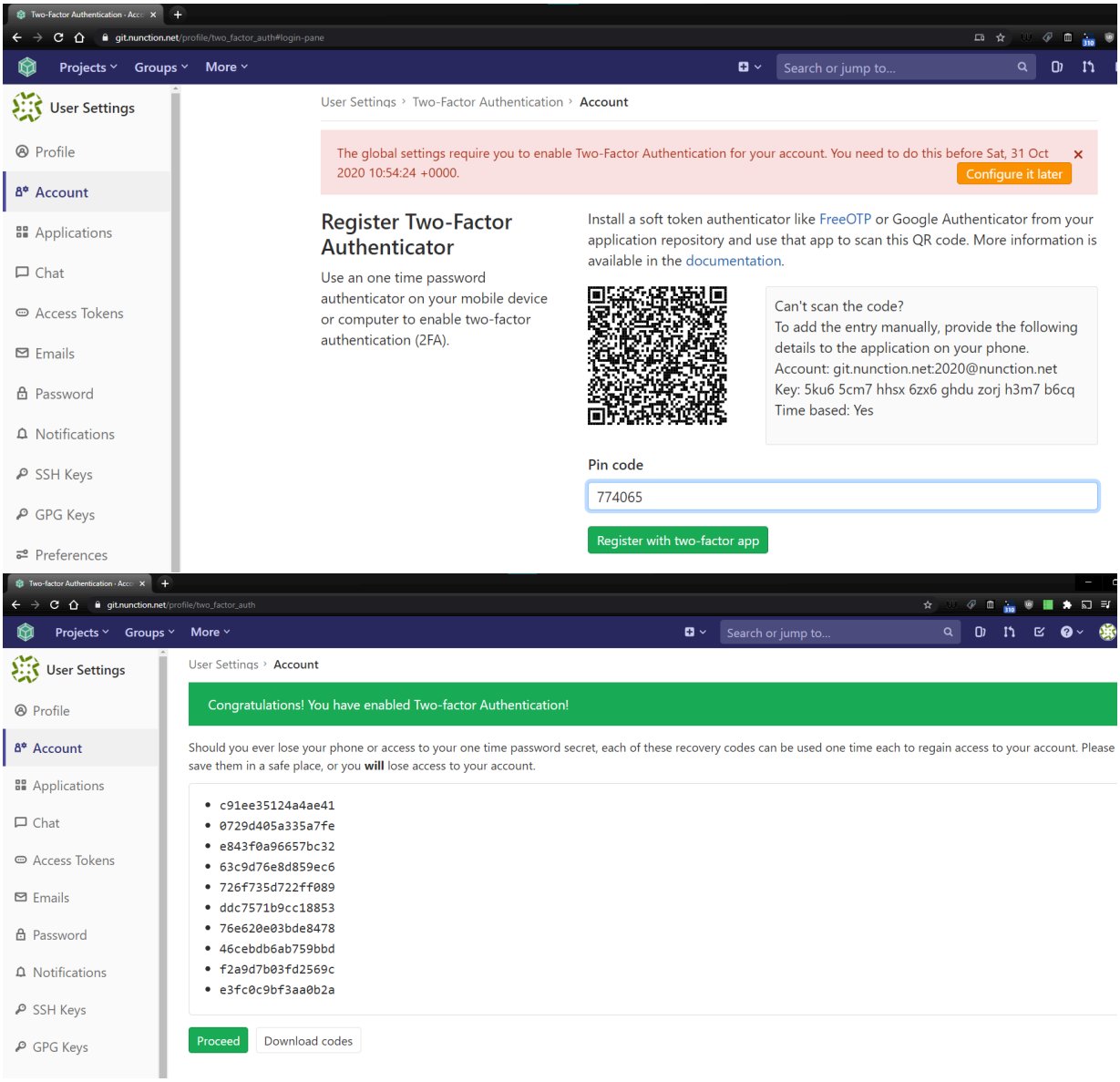

Abb. 5.13: Bestätigung des hinzugefügten Token auf Seiten des Anbieters (GitLab).

5.1 Prototyp

Zusätzlich zu der Softwarekomponente besteht die Anwendung aus einer Hardwarekomponente. Bei dieser Hardwarekomponente handelt es sich um einen Prototypen aufbauend auf der Raspberry Pi Plattform. Dieser Prototyp stellt in einem Paket ein Nutzerinterface, siehe Abbildung 5.14, in Form eines Displays und einem 4+1 Knöpfe D-Pad und Recheneinhat zur Verfügung.

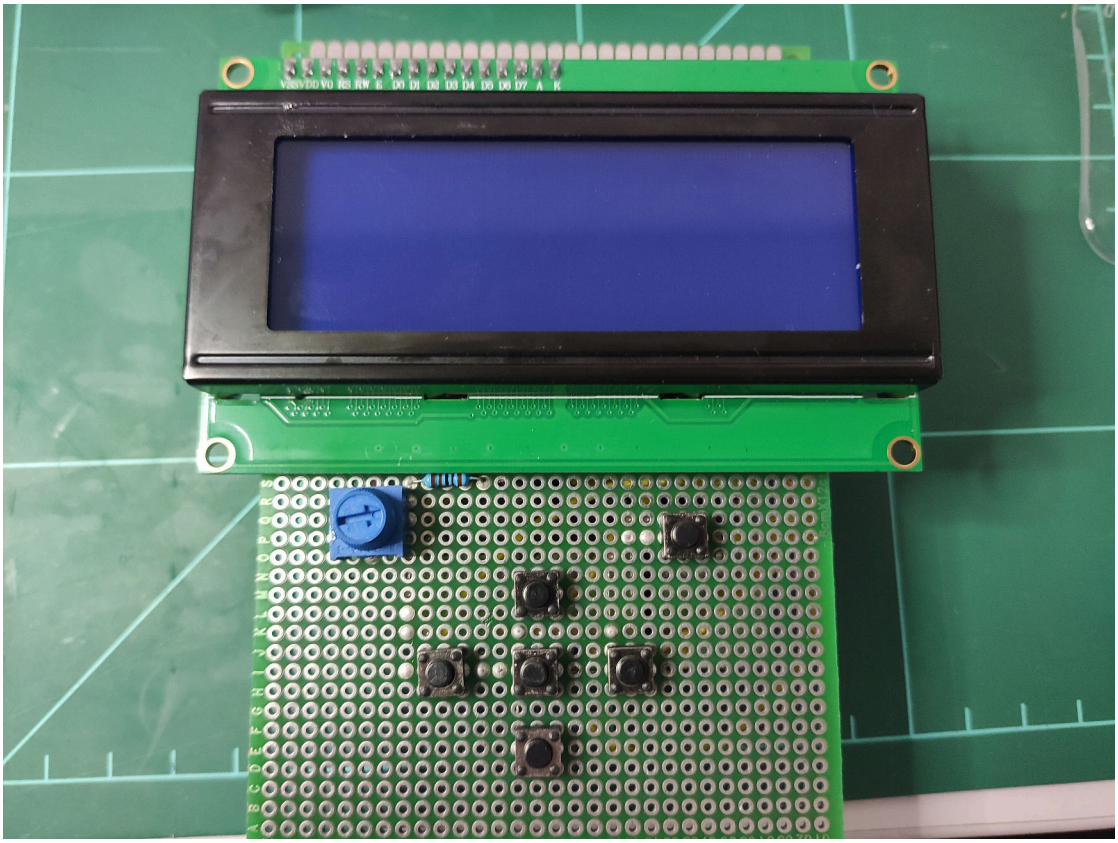

Abb. 5.14: Frontansicht des Hardwareprototypen.

Die Rückseite des Prototypen, siehe Abbildungen 5.15 und 5.16, wird zum Verbinden der verschiedenen Komponenten mit dem Raspberry Pi verwendet.

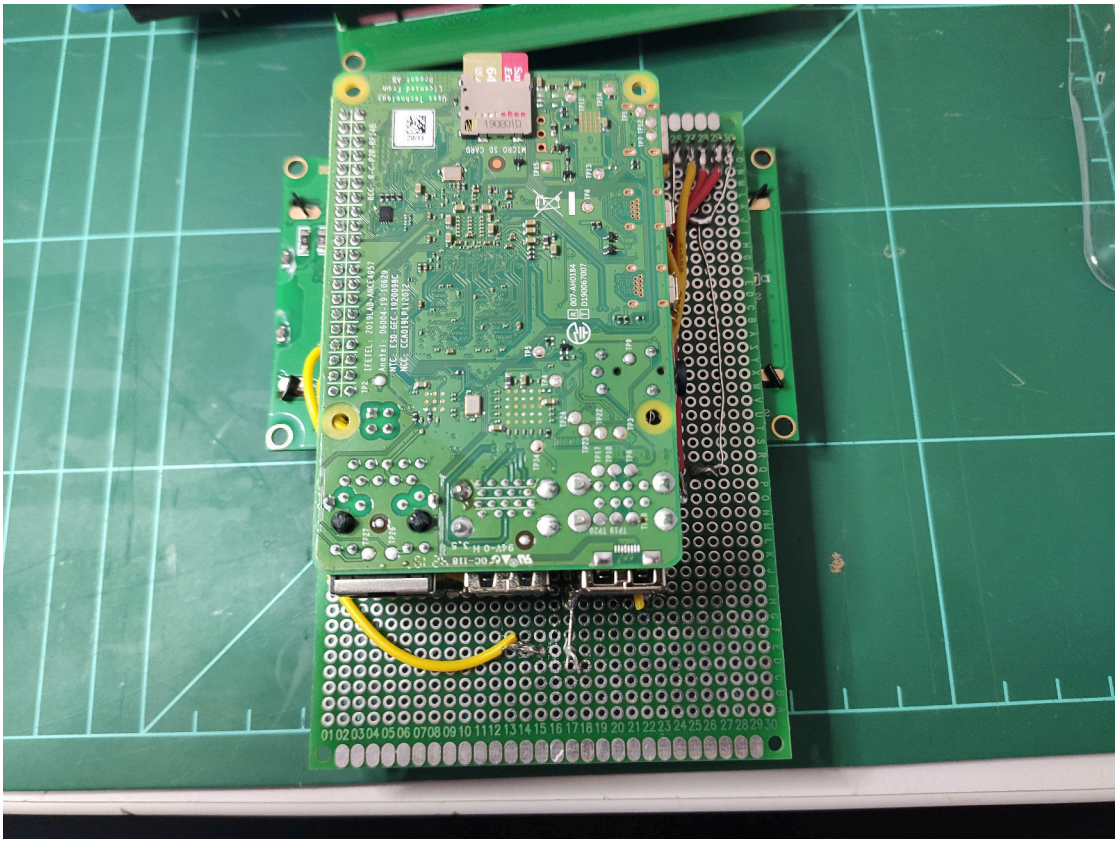

Abb. 5.15: Rückansicht des Hardwareprototypen.

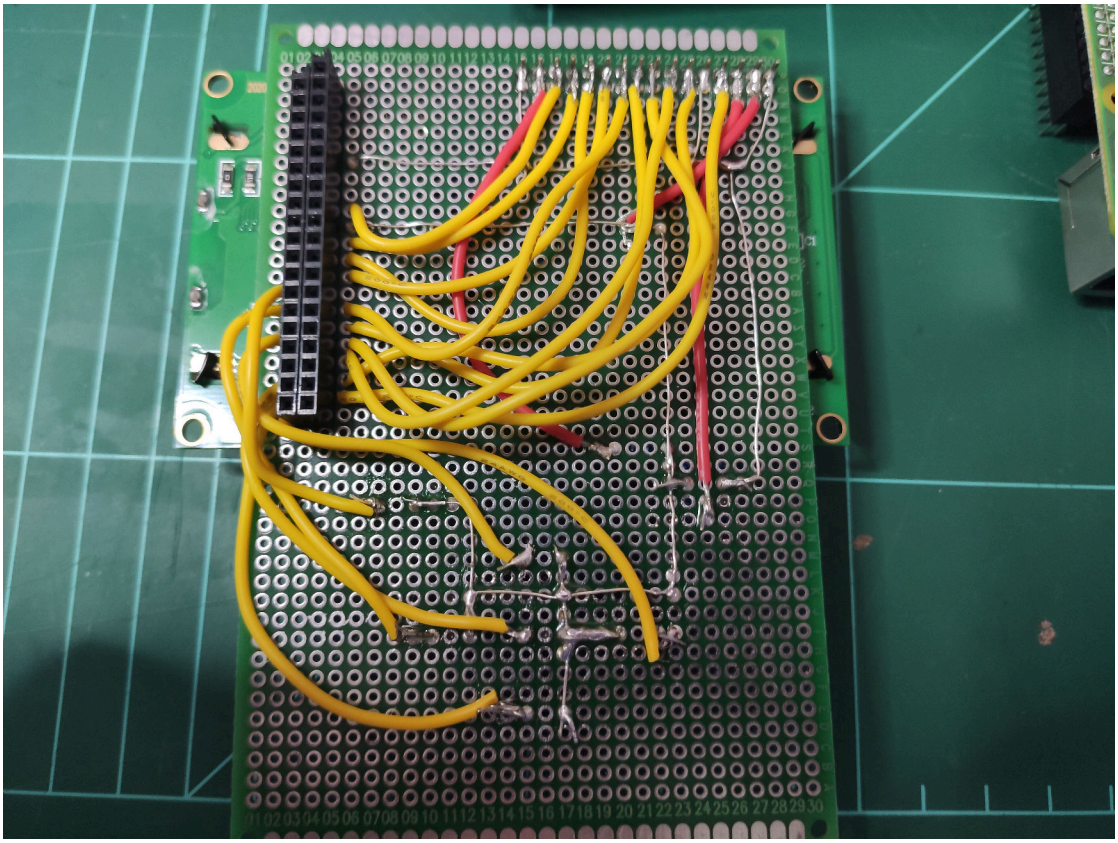

Abb. 5.16: Rückansicht des Hardwareprototypen mit entfernter Recheneinheit.

6

Sicherheitsanalyse

6.1 HOTP und TOTP

OTP wie HOTP und TOTP setzten verschiedene Sicherheitsanforderungen voraus. HOTP und TOTP verfügen über keinen eingebauten Mechanismus zur Verschlüsselung des generierten Passwortes. Daher muss das generierte Passwort mithilfe eines anderen Mechanismus, wie TLS, während dem Transport gesichert werden. Ebenso definieren weder HOTP noch TOTP einen Mechanismus zum Schutz des geteilten Geheimnis während dem Austausch zwischen Client und Server [8, 9].

6.1.1 Angriffe

In der Sicherheitsanalyse im Anhang der Algorithmendefinition von HOTP in RFC4226 wird die Schlussfolgerung getroffen, dass von den bestehenden Schwachstellen ein Brute-force Angriff die besten Chancen hat HOTP zu umgehen. Um die anfällig gegen Brute-force Angriffe zu mitigieren muss ein Verifikator diese erkennen und blockieren. Dies kann erzielt werden durch das Limitieren der Loginversuche [8, 9]. Das Auftreten von neuen praktikablen Angriffen gegen SHA-1, wie SHAttered [12], wirkt sich jedoch nicht negativ auf die kryptografischen Eigenschaften von HMCA-SHA-1 in Verwendung für HOTP und TOTP, wie in RFC4226 beschrieben, aus [8].

6.2 Anwendung

Die Anwendung verwendet bei TLS die von der Dotnet Laufzeitumgebung bereitgestellte TLS Implementierung in Form der Klasse SSLStream. Dies bedeutet, dass die meisten Laufzeitumgebungen openSSL zur Implementierung von TLS verwenden. Andere Laufzeitumgebungen verwenden ähnliche kryptografische Bibliotheken wie WinCrypt, axTLS oder BearSSL. Dies bezieht sich ebenso auf die verwendeten Entropiequellen für kryptografische Operationen, auf Windows und Linux Plattformen werden die vom Betriebssystem bereitgestellten Entropiequellen verwendet. Die temporären x509 Zertifikate und deren ECC Schlüssel werden ebenso von der Laufzeitumgebung ohne direkten Einfluss des Entwicklers generiert. Dies

reduziert die Gefahr, dass der Entwickler Fehler bei der Implementierung oder bei den kryptografischen Einstellungen macht.

6.2.1 Lokales Schnittstelle

Die lokale Schnittstelle bezieht ihre Sicherheit primär dadurch, dass diese an ein physikalisches Objekt, die Hardware, gebunden ist. Diese stellt bis auf den Zugriff durch die entfernte Schnittstelle ein Air-Gapped System dar. Erhält ein Angreifer zugriff zu dieser Hardware hat er volle Kontrolle über die lokale Schnittstelle. Um dies zu mitigieren ist in einer zukünftigen Version eine in Anwendungsverschlüsselung der Daten geplant. Dies benötigt jedoch, das ein Benutzer bei jedem Systemstart ein Masterpasswort eingibt. Dieses Masterpasswort muss genug Entropie aufweisen um die gespeicherten Parameter zu schützen. Dies stellt jedoch einen signifikanten Aufwand auf Seiten des Benutzers dar, da die Hardware über keine Tastatur verfügt und somit alle Eingaben über das D-Pad erfolgen müssen. Ein weiterer möglicher Ansatz zum Schutz der Parameter wäre die Verwendung eines Hardware Trusted Platform Modules in Kombination eines PINs und Full-Disk Encryption. Dies ist jedoch kein vom Linuxkernel nativ unterstütztes Szenario und benötigt Entwicklungsaufwand seitens des Linuxkernels.

Auf der Raspberry Pi Plattform läuft die Anwendung nicht direkt auf der Hardware sondern als Dienst in dem auf Debian basierenden Betriebssystem Raspbian. Um mögliche Schwachstellen von Seiten des Betriebssystems zu midigieren verfügt dieses lediglich einen minimalen Satz an installierten Anwendungen und Diensten. Des weiteren werden Standard Linux Hardening-Techniken verwendet um das System abzuschotten. Das System selbst stellt keine Verbindung ins Internet oder anderes lokales Netzwerk her. Anstelle dessen spannt das System mithilfe der eingebauten Hardware ein eigenes WiFi Netzwerk auf über welches sich die entfernte Schnittstelle verbinden kann.

6.2.2 Entfernte Schnittstelle

Die entfernte Schnittstelle baut für ihre Sicherheit auf zwei grundlegenden Aspekten auf. Zu einem muss die Verbindung zu der Anwendung vom Benutzer jedes Mal auf der Hardware bestätigt werden. Zur Identifizierung wird ein Fingerabdruck eines TLS Zertifikates von der entfernten Schnittstelle verwendet. Dieses Zertifikat wird mit jedem Verbindungsaufbau ausgetauscht. Dadurch, dass ein Benutzer auf Seiten der Hardware die Verbindung bestätigen muss und es bei Zertifikaten schwer ist, einen kollidierenden Fingerabdruck zu erzeugen wird somit eine beidseitige Authentifizierung hergestellt. Neben der Sicherung der Sitzung und des Transportweges mittels TLS ist die entfernte Schnittstelle weiterhin in ihrer Funktionalität eingeschränkt. Ein Angreifer, der die entfernte Schnittstelle übernimmt oder sich als diese ausgibt erlangt keinen Zugriff auf die geteilten Geheimnisse oder daraus generierten Codes da diese bei Verwendung der entfernten Schnittstelle weiterhin nur auf der Hardware selbst angezeigt werden. Ein Angreifer der die entfernte Schnittstelle übernehmen kann ist somit lediglich in der Lage einen Denial

Of Service durch löschen oder editieren nicht kritischer Parameter durchzuführen. Ein Angreifer der Zugriff auf das System hat welches zum Hinzufügen eines neuen Token genutzt wird stellt ebenso aus Sicht der Anwendung kein Risiko dar, da dieser bereits vor Interaktion mit der Anwendung das geteilte Geheimnis abfangen kann.

7

Schlussfolgerung

Der Prototyp der Anwendung erfüllt die minimal notwendigen Anforderungen um als ein Security-Token für HOTP und TOTP verwendet zu werden. Der Prototyp ist jedoch lediglich geeignet als Übergangslösung bis das HOTP und TOTP durch modernere Mechanismen wie FIDO, FIDO2 oder SQRL abgelöst werden. Diese Übergangsperiode kann potentiell jedoch noch einige Zeit anhalten da zur Zeit der Ausarbeitung noch nicht alle Seiten MFA, oder andere MFA Verfahren neben SMS unterstützen. Ebenso ist anzumerken dass TOTP gegenüber HOTP, aufgrund der Synchronisierungsprobleme und veralteten kryptografischen Funktionen in HOTP, zu bevorzugen ist.

Probleme mit der primären Plattform

Die Meadow Plattform ist zwar die primäre Plattform für die konzipierte Anwendung, diese war jedoch zur Zeit der Ausarbeitung aus logistischen Gründen nicht erhaltbar. Somit wurde die Anwendung für die sekundäre Plattform entwickelt. Die Anpassung der Anwendung auf die primäre Plattform ist jedoch durch die Verwendung der Dotnet Laufzeitumgebung ein triviales zukünftiges Unterfangen. Da lediglich die Schnittstelle zur Bereitstellung der GPIOs auf die neue Plattform angepasst werden muss.

7.1 Ausblick

Eine zukünftige Version der Hardware könnte zur besseren Handhabung mit einem Gehäuse ausgestattet werden. Des weiteren kann der Raspberry Pi durch den Meadow abgelöst werden was die gesamte Größe der Hardware reduziert. Auch kann anstelle von einer Breadboard-Style Prototypen Platine eine Platine speziell für die Hardware entworfen und produziert werden. Diese Platine kann die notwendigen Komponenten direkt in ein Surface-Mount-Design übernehmen. Um die Benutzerfreundlichkeit der Anwendung zu verbessern kann das UI sowohl lokal als auch im Browser überarbeitet werden.

Eine weitere notwendige Verbesserung des Systems ist ein Update-Mechanismus. Da das System über keine Verbindung in ein Netzwerk verfügt muss ein Update-Mechanismus über die entfernte Schnittstelle implementiert werden. Um die Sicherheit der Updates zu gewährleisten müssen diese mit einer Digitalen Signatur

ausgestattet werden. Jedoch ist ein solcher Update-Mechanismus erst umsetzbar sobald die primäre Plattform, Meadow, verwendet wird da auf der sekundären Plattform, Raspberry Pi, ein Over The Air Update-Mechanismus aufgrund des komplexen Betriebssystems nicht einfach implementieren lässt.

Um das Problem der Fingerabdrucklänge zu umgehen kann eine zukünftige Version der Anwendung auf TLS mit PSK setzen, wobei der PSK auf der Hardware bei jeder Verbindung neu generiert wird. Ein Benutzer muss den PSK von der Hardware ablesen und in der entfernten Schnittstelle eintragen und somit eine beidseitige Authentifizierung durchführen. Jedoch würde eine Zertifikat basierte Authentifizierung erlauben die Zertifikate für eine bestimmte Zeit wiederzuverwenden und somit eine Re-Authentifizierung bei jeder Verbindung zu vermeiden.

8

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Ausarbeitung mit dem Titel

"Entwicklung eines Open Source Hash und Time basierten One-Time-Password Hardware Token"

selbstständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Trier, den 30. Oktober 2020

A

Attachments

Elektrische Anhänge befinden sich in der mitgelieferten Datei 'attachments.zip' und werden mit ihrem relativen Pfad zum Archiv Wurzelpfad angegeben.

Projektübersicht

- 'Nunction.MOTP/src/Nunction.IO.CharacterLCD' : Projekt zur Implementierung des Hardwaretreibers des LCD
- 'Nunction.MOTP/src/Nunction.IO.Primitives' : Projekt zur Definition von abstrakten Hardware Schnittstellen
- 'Nunction.MOTP/src/Nunction.Io.Sysfs' : Projekt zur Implementierung der GPIO Hardwareschnittstelle des RPI
- 'Nunction.MOTP/src/Nunction.MOTP.Common' : Projekt zur Implementierung von geteilten Transferobjekten.
- 'Nunction.MOTP/src/Nunction.MOTP.Core' : Projekt zur Implementierung der Erstellung, Verwaltung und Persistenz von Token, sowie der Grafikbibliothek
- 'Nunction.MOTP/src/Nunction.MOTP.UI.Console' : Projekt zur Implementierung der Schnittstellen (local and remote server)
- 'Nunction.MOTP/src/Nunction.MOTP.UI.Web' : Projekt zur Implementierung der client seite der entfernten Schnittstelle
- 'Nunction.MOTP/src/Nunction.Networking.SXNP' : Projekt zur Implementierung der Netzwerkbibliothek
- 'Nunction.MOTP/src/SysfsTest' : Projekt für manuelle Tests (Hardware)
- 'Nunction.MOTP/test/Nunction.MOTP.Core.Tests' : Projekt für autom. Unit Tests

Übersicht über Datei

- 'Nunction.MOTP/src/Nunction.MOTP.Core/OTP/HOTP/HOTPProvider.cs' : Implementierung von HOTP
- 'Nunction.MOTP/src/Nunction.MOTP.Core/OTP/TOTP/TOTPProvider.cs' : Implementierung von TOTP
- 'Nunction.MOTP/src/Nunction.MOTP.Core/UI/Common/DisplayManager.cs' : Implementierung der Verwaltung von UI Elementen

- 'Nunction.MOTP/src/Nunction.MOTP.UI.Console/LocalInterface.cs' : Implementierung der lokalen Schnittstelle
- 'Nunction.MOTP/src/Nunction.MOTP.UI.Console/RemoteInterface.cs' : Implementierung des Servers der entfernten Schnittstelle
- 'Nunction.MOTP/src/Nunction.Networking.SXNP/Common/Session.cs' : Implementierung der Verbindung der Netzwerkbibliothek
- 'Nunction.MOTP/src/Nunction.MOTP.UI.Web/Pages/Index.razor : Implementierung des Clients der entfernten Schnittstelle

Literaturverzeichnis

1. Microsoft. Introduction to asp.net core blazor.
2. Paul A. Grassi, Michael E. Garcia, and James L. Fenton. Nist special publication 800-63 digital identity guidelines. Special Publication 800-63, National Institute of Standards and Technology, Jun 2017.
3. Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication. RFC 2104, RFC Editor, February 1997. `http://www.rfc-editor.org/rfc/rfc2104.txt`.
4. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, RFC Editor, May 2008. `http://www.rfc-editor.org/rfc/rfc5280.txt`.
5. T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008.
6. Philip Japikse. *CSharp 6.0 and the .NET 4.6 Framework*. Apress, 2017.
7. Paul A. Grassi, James L. Fenton, Elaine M. Newton, Ray A. Perlner, Andrew R. Regenscheid, William E. Burr, and Justin P. Richer. Nist special publication 800-63b digital identity guidelines. Special Publication 800-63b, National Institute of Standards and Technology, Jun 2017.
8. D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. Hotp: An hmac-based one-time password algorithm. RFC 4226, RFC Editor, December 2005.
9. D. M'Raihi, S. Machani, M. Pei, and J. Rydell. Totp: Time-based one-time password algorithm. RFC 6238, RFC Editor, May 2011.
10. Hitachi, Ltd. *HD44780U (LCD-II)*, 0.0 edition, 1998.
11. Maxim Integrated Products, Inc. *DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal*, 0.0 edition, 2015.
12. Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. Technical report, 2017.