ST. PÖLTEN UNIVERSITY
OF APPLIED SCIENCES

Informatik
& Security

fh
st.pölten

Sensorseitiges Wasserzeichen als kontinuierliche Authentifizierungsmethode

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

eingereicht von

Scherscher Gabriel

is211841

im Rahmen des
Studienganges Information Security an der Fachhochschule St. Pölten

Betreuung
Betreuer/in: Dipl.-Ing. Dr. Henri Ruotsalainen
Mitwirkung: -

St. Pölten, 9. April 2024 _____ _____

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Fachhochschule St. Pölten GmbH, Campus-Platz 1, A-3100 St. Pölten,T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E: office@fhstp.ac.at, I: www.fhstp.ac.at

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich das Thema dieser Arbeit bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Ort, Datum *Unterschrift*

Kurzfassung

Angesichts des rasanten Wachstums des Marktes für Hausautomatisierungssysteme (HAS) und Internet-of-Things (IoT)-Geräte und der zunehmenden Ausstattung von Haushalten mit diesen Technologien, steigt auch die Attraktivität dieser Systeme für potenzielle Angreifer. Diese Entwicklung unterstreicht die dringende Notwendigkeit, die Sicherheit dieser Geräte zu erhöhen, die derzeit hauptsächlich durch Authentifizierungsmechanismen mit geheimen Schlüsseln geschützt werden. Diese Methoden bieten jedoch keinen Schutz gegen die Manipulation der analogen Schnittstellen, was eine signifikante Sicherheitslücke darstellt. Vor diesem Hintergrund ist es das Ziel dieser Arbeit, aufbauend auf den Vorarbeiten von Ruotsalainen et al. das Konzept des sensorseitigen Wasserzeichens zur kontinuierlichen Authentifizierung zu entwickeln und zu implementieren, um HAS und IoT-Geräte vor physischer Manipulation zu schützen [1].

Der methodische Ansatz umfasste die Implementierung und Anpassung des Wasserzeichenkonzepts für eine kontinuierliche Authentifizierung. Durch die schrittweise Erprobung der einzelnen Komponenten des Konzepts, die Erweiterung um ein HAS-Gateway und die anschließende Integration in ein Gesamtkonzept konnte die Umsetzung erfolgreich realisiert werden.

Umfangreiche Tests wurden durchgeführt, um die Leistungsfähigkeit des Systems sowohl unter Normalbedingungen als auch unter spezifischen Angriffsszenarien zu evaluieren. Diese Tests haben gezeigt, dass das System im Normalbetrieb mit einer Bit Error Rate (BER) nahe Null Prozent und einer daraus resultierenden False Negative Rate (FNR) von 0% bei Raumtemperatur eine hervorragende Performance aufweist. In den Angriffsszenarien bestätigte eine BER von ca. 50% die erfolgreiche Erkennung von Manipulationsversuchen, was die Effektivität des entwickelten Authentifizierungssystems unterstreicht. Herausforderungen traten bei extrem niedrigen Temperaturen auf, wo ein signifikanter Anstieg der BER beobachtet wurde, was die Notwendigkeit unterstreicht, das System für ein breiteres Spektrum von Umgebungsbedingungen zu optimieren.

Zusammenfassend liefert diese Arbeit einen wichtigen Beitrag zur Verbesserung der Sicherheit von HAS und IoT-Geräten durch die Implementierung eines innovativen Authentifizierungskonzepts basierend auf sensorseitigen Wasserzeichen. Zukünftige Forschungsarbeiten könnten sich darauf konzentrieren, das System für extremere Bedingungen zu optimieren und das Konzept auf ein breiteres Spektrum von Anwendungsfällen auszuweiten, um die Sicherheit weiter zu erhöhen.

Abstract

With the rapid growth of the market for Home Automation Systems (HAS) and Internet of Things (IoT) devices and the increasing adoption of these technologies in the private homes, these systems have become attractive targets for potential attackers. This development highlights the urgent need to improve the security of these devices, which are currently protected primarily by secret key authentication mechanisms. However, these methods do not protect against the manipulation of analogue interfaces, which represents a significant security vulnerability. Against this background, this work aims to develop and implement the concept of sensor-side watermarking for continuous authentication, based on the preliminary work of Ruotsalainen et al. to protect HAS and IoT devices from physical tampering. [1]

The methodological approach involved the implementation and adaptation of the continuous authentication watermarking concept. By sequentially testing the individual components of the concept, extending it with a HAS gateway and subsequently integrating it into a complete system, the implementation was successfully realised.

Extensive tests were conducted to evaluate the performance of the system under normal conditions and specific attack scenarios. These tests showed that the system has excellent performance in normal operation, with a Bit Error Rate (BER) close to zero percent and a resulting False Negative Rate (FNR) of zero percent at room temperature. In the attack scenarios, a BER of around 50% confirmed the successful detection of tampering attempts, underlining the effectiveness of the authentication system developed. Challenges were observed at extremely low temperatures, where a significant increase in BER was observed, highlighting the need to optimise the system for a wider range of environmental conditions.

In conclusion, this work makes a significant contribution to improving the security of HAS and IoT devices through the implementation of an innovative authentication concept based on sensor-side watermarks. Future research could focus on optimising the system for more extreme conditions and extending the concept to a wider range of applications to further enhance security.

Inhaltsverzeichnis

1 Einleitung

1.1 Relevanz des Themas

Der Markt für Hausautomatisierungssysteme (HAS) und Internet-of-Things-Geräte (IoT) ist in den letzten Jahren stark gewachsen. Immer mehr Haushalte sind mit solchen Geräten ausgestattet, was sie zu einem attraktiven Ziel für Angreifer macht. Aufgrund dieser steigenden Relevanz wird auch die Absicherung von HAS und IoT-Geräten immer wichtiger. Derzeit ist die Sicherheit von HAS und IoT-Geräten jedoch sehr mangelhaft, da sie meist nur durch Authentifizierung mittels geheimer Schlüssel geschützt sind.[2] Ungeschützt bleiben dabei meist die analogen Schnittstellen der Geräte. Aufgrund der ansteigenden Bedrohung wird es nötig diese Sensore mit kontinuierlichen Überprüfungen zu schützen, um größere Schäden durch physische Manipulation zu verhindern.

1.2 Problemstellung

Aktuell sind HAS und IoT-Geräte meist nur mittels Authentifizierung auf Basis von geheimen Schlüsseln, die beim Beitritt ins Netzwerk ausgetauscht werden, geschützt. [2] Diese Art der Authentifizierung schützt jedoch nicht gegen die Manipulation von Analogen Schnittstellen. Diese Lücke wurde in *Watermarking Based Sensor Attack Detection in Home Automation Systems* und anderen Arbeiten versucht zu schließen, indem mittels Anpassungen der Versorgungsspannung eine Art Wasserzeichen in die Ausgangsspannung eingefügt wird. Dieses Wasserzeichen erlaubt es, verschiedene Arten von physischen Angriffen, wie zum Beispiel die Erdung des Analog-Digital-Koverters (ADC), zu erkennen. [1], [3] Als nächster Schritt dieser Technologie ist es nötig, basierend auf der Vorarbeit aufbauend, eine kontinuierliche Überprüfung an einem zentralen Gateway zu errichten, welche es noch schwerer für Angreifer macht die einzelnen analogen Signale unbemerkt zu verändern.

1.3 Ziele und Forschungsfragen

Ziel der Arbeit ist es ein komplettes Konzept für die kontinuierliche Überprüfung von sensorseitigen Wasserzeichen zu entwerfen und umzusetzen. Dabei soll die Funktionsfähigkeit im Normalbetrieb wie auch im Angriffsfall getestet und analysiert werden, um die Leistung des Konzeptes evaluieren und mögliche Angriffsszenarien identifizieren zu können. Die Erkenntnisse aus diesen Tests sollen also dazu dienen, das Verständnis für die Wirksamkeit und Robustheit der sensorseitigen Wasserzeichenmethode zu vertiefen und Möglichkeiten zur Verbesserung aufzuzeigen.

1.3.1 Forschungsfragen

- Wie könnte ein umfassendes Konzept für die kontinuierliche Überprüfung von sensorseitigen Wasserzeichen entwickelt und implementiert werden?
- Wie verhalten sich die False Positive und False Negative Raten des entwickelten Authentifizierungskonzeptes im Normalbetrieb sowie in den Angriffsszenarien Grounding Attack und Voltage Injection und welche Maßnahmen können ergriffen werden, um diese Raten zu minimieren und die Zuverlässigkeit des Systems zu verbessern?
- Welche unerwarteten Schwachstellen und potenziellen Angriffsszenarien wurden während der Implementierung und Testphase des sensorseitigen Wasserzeichenüberprüfungssystems entdeckt, und wie können diese Schwachstellen effektiv adressiert werden, um die Sicherheit und Robustheit des Systems zu verbessern?

Die Untersuchung dieser Forschungsfragen soll dazu beitragen, ein tieferes Verständnis für die Funktionsweise und Effektivität von sensorseitigen Wasserzeichen als kontinuierliche Authentifizierungsmethode zu entwickeln und mögliche Wege zur Weiterentwicklung dieser Technologie aufzuzeigen.

1.4 Abgrenzung zu anderen Arbeiten

Diese Diplomarbeit setzt neue Akzente im Bereich der sensorseitigen Wasserzeichen durch die Einführung einer kontinuierlichen Verifikation und die Implementierung eines HAS-Gateways als zentralen Authentifizierungspunkt. Die wesentlichen Unterscheidungsmerkmale zu vorherigen Arbeiten umfassen:

- **Kontinuierliche Überprüfung:** Die Implementierung einer kontinuierlichen sensorseitigen Überprüfung von Wasserzeichen stellt einen aufkommenden Ansatz dar. Diese Methode ermöglicht es, Sicherheitsbedrohungen in Echtzeit zu erkennen und darauf zu reagieren, wodurch die Sicherheit von HAS

und IoT-Geräten erheblich verbessert werden kann.

- **Verwendung eines HAS-Gateways:** Im Gegensatz zu Ansätzen, die keine zentrale Überprüfung vorsehen, integriert die vorliegende Arbeit ein HAS-Gateway als zentralen Authentifizierungspunkt für HAS-Clients. Dies ermöglicht eine effektivere und effizientere Sicherheitsüberwachung.
- **Anpassung an spezifische Hardware:** Die Arbeit demonstriert die Übertragbarkeit und Flexibilität des von Ruotsalainen u. a. entwickelten Konzepts durch die Verwendung eines anderen Mikrocontrollers als den bisher in der Forschung [1] verwendeten. Diese Anpassungsfähigkeit zeigt die praktische Anwendbarkeit über verschiedene Technologieplattformen hinweg
- **Zusätzliche Tests zur Umgehung der Authentifizierung:** Durch die Durchführung eines spezifischen Tests, die darauf abzielen, die Authentifizierung erfolgreich zu umgehen, bietet die Arbeit neue Einblicke in potenzielle Sicherheitsschwachstellen und trägt zum tieferen Verständnis der Robustheit von sensorseitigen Wasserzeichen bei.

1.5 Struktur der Arbeit

Dieses Dokument ist in mehrere Teile gegliedert: Kapitel 1 dient als grundlegende Einführung in das bearbeitete Thema, durch die Beschreibung von Problemen, Herausforderungen und der Motivation der Arbeit. Kapitel 2 listet verwandte Arbeiten auf. Kapitel 3 beschreibt die Voraussetzungen, die Vorgehensweise und die wichtigsten Arbeitsschritte der Arbeit. Kapitel 4 definiert verschiedene Angriffsszenarien, sowie die Umsetzung dieser. Kapitel 5 berichtet über die Ergebnisse der durchgeführten Tests im Normalbetrieb, sowie im Angriffsszenario. Kapitel 6 diskutiert die erhaltenen Ergebnisse . Kapitel 7 fasst die Arbeit zusammen.

2 Stand der Forschung

Die Integration von Wasserzeichen in sensorseitige Datenströme bietet einen innovativen Ansatz zur Erhöhung der Sicherheit in HAS und IoT-Geräten. Diese Methode adressiert speziell die Herausforderung, die Authentizität von Sensordaten in Echtzeit zu verifizieren, und schützt gegen eine Vielzahl von Angriffen, einschließlich physischer Manipulation und nichtinvasiver Sensor-Spoofing-Angriffe [1].

2.1 Erkennung von Sensor-Angriffen

In den letzten Jahrzehnten war nicht nur eine Zunahme von Sensoren in eingebetteten und cyber-physischen Systemen (CPS) zu beobachten, sondern auch eine Evolution der Angriffsmethoden auf diese kritischen Komponenten. Sensoren, die als Brücke zwischen der physischen Welt und digitalen Systemen dienen, sind zunehmend zum Ziel von Angreifern geworden, deren Ziel es ist, die Integrität und Authentizität der Daten zu gefährden. Diese Angriffe reichen von physischen Manipulationen bis hin zu ausgeklügelten, nichtinvasiven Spoofing-Angriffen, bei denen durch gefälschte Signale Fehlinterpretationen und Fehlentscheidungen übergeordneter Systeme provoziert werden[4].

Um solchen Bedrohungen effektiv begegnen zu können, ist es entscheidend, robuste Erkennungsmechanismen zu entwickeln, die in der Lage sind, ungewöhnliche Muster oder Anomalien in den Sensordaten zu identifizieren. Diese Mechanismen müssen fortgeschrittene Techniken wie maschinelles Lernen und statistische Analysemethoden verwenden, um auch subtile Spoofing-Versuche erkennen zu können. Die Herausforderung liegt nicht nur in der Genauigkeit der Erkennung, sondern auch in der Minimierung von Fehlalarmen, die den normalen Betrieb des Systems stören könnten[4].

2.2 Nicht-invasive Angriffe

Nicht-invasive Sensor-Spoofing-Angriffe stellen eine besonders heimtückische Form der Bedrohung dar, da sie ohne physischen Zugriff auf die Zielgeräte durchgeführt werden können. Angreifer verwenden exter-

ne Quellen, um gefälschte oder manipulierte Signale zu erzeugen, die dann von den Sensoren als legitime Messwerte interpretiert werden. Die negativen Folgen solcher Angriffe sind vielfältig und reichen von der Fehlfunktion sicherheitskritischer Anwendungen bis hin zum kompletten Systemausfall.

Die Entwicklung von Sicherheitsmechanismen, die über traditionelle Ansätze hinausgehen, ist notwendig, um die Integrität und Authentizität der Sensordaten unter solchen Bedrohungen zu gewährleisten. Dies erfordert einen mehrschichtigen Sicherheitsansatz, der sowohl auf der Ebene der einzelnen Sensoren als auch auf der Ebene der Systemarchitektur ansetzt. Mögliche Strategien sind die Einführung von Redundanzen in Sensornetzwerken, der Einsatz von Physical Unknown Functions (PUFs) zur Authentifizierung von Sensordaten und die Implementierung von Ende-zu-Ende-Verschlüsselungsmechanismen zur Sicherung der Datenübertragung zwischen Sensoren und übergeordneten Systemen. Darüber hinaus kann die kontinuierliche Überwachung und Analyse des Netzwerkverkehrs dabei helfen, ungewöhnliche Aktivitäten frühzeitig zu erkennen und entsprechende Gegenmaßnahmen einzuleiten [4].

2.3 Resiliente Erkennung und Bewertung unter Cyber-Angriffen

Die zunehmende Integration von CPS in kritische Infrastrukturen unterstreicht die Bedeutung einer robusten Sicherheitsarchitektur, die sowohl physische als auch Cyber-Angriffe berücksichtigt. Insbesondere die Bedrohung durch False Data Injection (FDI)- und Jamming-Angriffe stellt eine signifikante Bedrohung für die Integrität und Zuverlässigkeit von CPS dar. In der Arbeit von Guan und Ge wird ein verteiltes System zur Angriffserkennung und sicheren Schätzung für vernetzte CPS vorgestellt, das FDI- und Jamming-Angriffen widerstehen kann. Durch die Verwendung eines drahtlosen Sensornetzwerks (WSN) zur Überwachung des Systems werden sowohl lokale zuverlässige Zustandsschätzungen als auch die Erkennung des FDI-Angriffs ermöglicht [5].

2.4 Herausforderungen und Lösungsansätze

Trotz des Potenzials, das sensorseitige Wasserzeichen bieten, stehen Forschung und Entwicklung vor einer Reihe von Herausforderungen. Dazu gehören die Erkennung subtiler Manipulationen von Sensordaten, die Integration von Wasserzeichen in bestehende Systemarchitekturen und die Sicherstellung, dass der Verifikationsprozess keine negativen Auswirkungen auf die Systemleistung hat [4], [6].

2.5 Forschungslücken

Während die Arbeit von Ruotsalainen et al. [1] sowie die Untersuchungen von Barua und Al Faruque [4] innovative Ansätze zur Absicherung von HAS und IoT-Geräten durch sensorseitige Wasserzeichen und die Erkennung von nichtinvasiven Sensor-Spoofing-Angriffen vorstellen, zeigen sie gleichzeitig signifikante Forschungslücken in diesen Bereichen auf. Eine zentrale Herausforderung besteht darin, die subtilen Manipulationen von Sensordaten, die durch ausgeklügelte Spoofing-Angriffe entstehen können, effektiv zu erkennen und zu verifizieren. Darüber hinaus sind die Integration dieser Wasserzeichentechniken in bestehende Systemarchitekturen und ihre Kompatibilität mit der vorhandenen Infrastruktur noch weitgehend unerforscht. Insbesondere die Skalierbarkeit und Anpassungsfähigkeit dieser Sicherheitsmaßnahmen in verschiedenen industriellen und kommerziellen Umgebungen stellen weiterhin offene Fragen dar.

2.6 Zukünftige Forschungsschwerpunkte

Zukünftige Forschung sollte sich auf die Entwicklung fortgeschrittener Algorithmen konzentrieren, die eine nahtlose Integration von Wasserzeichen in Sensordaten ermöglichen, ohne die Leistung oder Funktionalität der Geräte zu beeinträchtigen. Von Interesse ist auch die Schaffung von Mechanismen zur schnellen Erkennung und Reaktion auf Sicherheitsverletzungen, die durch manipulierte Sensordaten verursacht werden [1], [4], [6].

2.7 Wasserzeichenverfahren zur Datensicherung in drahtlosen Sensornetzwerken

Ein weiterer interessanter Ansatz zur Sicherung der Datenintegrität in drahtlosen Sensornetzwerken (WSN) verwendet Wasserzeichen nicht direkt auf den Sensordaten, sondern auf den Daten, die vom Client zum Server übertragen werden. Vandana Dhiman und Padmavati Khandnor diskutieren in ihrer Arbeit verschiedene Strategien zur Implementierung von Wasserzeichen in WSNs, um sowohl die Sicherheit als auch die Effizienz der Datenaggregation zu verbessern. Diese Methoden zielen darauf ab, den Datenverkehr durch die Reduzierung von Redundanz zu minimieren und gleichzeitig ein hohes Maß an Datenintegrität zu gewährleisten. Durch die Integration von Wasserzeichen in die Datenübertragung können WSNs sicherstellen, dass die übertragenen Informationen authentisch und unverändert sind, was für Anwendungen, in denen Entscheidungen auf der Grundlage dieser Daten getroffen werden, kritisch ist [7]. Der Einsatz solcher Technologien stellt eine wesentliche Erweiterung der konventionellen Sicherheitsmaßnahmen dar und unterstreicht

die Notwendigkeit, umfassende Sicherheitslösungen zu entwickeln, die über die physische Sicherheit der Sensoren hinausgehen.

2.8 Sicherheitsherausforderungen von IoT-fähigen Sensoren in industriellen Umgebungen

Die Integration von IoT-fähigen Sensoren in industrielle Automatisierungssysteme bringt spezifische Sicherheitsherausforderungen mit sich, wie Thilo Sauter und Albert Treytl aufzeigen. IoT-Geräte, die zunehmend in kritischen Infrastrukturen und Produktionsanlagen eingesetzt werden, eröffnen neue Angriffsvektoren, die potenziell für schädliche Aktionen oder den Diebstahl sensibler Daten ausgenutzt werden können. Die Autoren betonen, wie wichtig es ist, das klassische Konzept der Verteidigungstiefe zu überdenken und anzupassen, um den Bedrohungen durch IoT-Geräte effektiv begegnen zu können. Eine besondere Herausforderung besteht darin, dass viele IoT-Geräte auf einfache Integration und Handhabung ausgelegt sind, was oft zu Lasten der Sicherheit geht. Dies erfordert neue Ansätze in der Sicherheitsarchitektur, die sowohl die physische als auch die cyber-technische Ebene der IoT-Geräte abdecken [8].

2.9 Physikalische Authentifizierung von Kontrollsystemen durch Wasserzeichen

Forschungsarbeiten im Bereich der physischen Authentifizierung von Steuerungssystemen, wie sie von Yilin Mo, Sean Weerakkody und Bruno Sinopoli durchgeführt wurden, zeigen, dass Wasserzeichen effektiv eingesetzt werden können, um gefälschte Sensorausgaben zu erkennen. Diese Arbeiten erweitern den Anwendungsbereich von Wasserzeichentechnologien über den Schutz digitaler Medien hinaus auf die Gewährleistung von Integrität und Authentizität in Steuerungssystemen. Durch das Einbringen von Wasserzeichen in Steuersignale können Systeme so gestaltet werden, dass sie Manipulationen von Sensorausgängen erkennen können, was eine kritische Sicherheitsfunktion in vielen industriellen und kritischen Infrastrukturanwendungen darstellt. Derartige Technologien ermöglichen die effektive Erkennung und Abwehr von Angriffen, die auf die Täuschung oder Sabotage von Sensoren abzielen, und stellen somit einen wichtigen Schritt zur Gewährleistung der Zuverlässigkeit und Sicherheit von cyber-physischen Systemen dar [9].

3 Methodik

In diesem Kapitel werden sämtliche Schritte für die kontinuierliche sensorbasierte Überprüfung von Wasserzeichen ausführlich beschrieben. Ziel ist es, die Nachvollziehbarkeit der Arbeit und die Überprüfbarkeit der erzielten Ergebnisse zu gewährleisten. Daher werden alle durchgeführten Schritte genau so dokumentiert, wie sie im Verlauf dieser Forschungsarbeit durchgeführt wurden.

3.1 Verwendete Hardware

Im Zuge dieses Unterkapitels wird die, für die Durchführung der Arbeit benötigte Hardware beschrieben.

3.1.1 Mikrocontroller ESP32 Wroom 32 Dev Kit

Der ESP32 ist einer der am weitest verbreiteten Mikrocontroller in der IoT-Entwicklung. Er zeichnet sich durch seine Robustheit und hohe Leistungsfähigkeit aus. Als hoch integrierter Mikrocontroller wurde er speziell für den Einsatz mit verschiedenen Kommunikationstechnologien entwickelt. Aufgrund seines breiten Anwendungsgebietes findet er sich in einer Vielzahl von Hobbyprojekten bis hin zu kommerziellen Produkten wieder.[10]
ESP32-WROOM-32 Spezifikationen:

- Prozessor: Tensilica Xtensa LX6 Mikroprozessor mit Dual-Core-Architektur und einer Taktfrequenz von bis zu 240MHz.
- Konnektivität: Integrierte Wi-Fi(02.11b/g/n)- und Bluetooth-Funktionalität für diverse Kommunikationsbedürfnisse.
- Speicher: 512 KB internes SRAM, mit der Möglichkeit zur Erweiterung durch externe Speicher.
- Analog-Digital-Wandler, mit einer 12 bit Auflösung.
- Digital-Analog-Wandler, mit einer 8 bit Auflösung.
- Entwicklungsumgebung: Unterstützung durch die Arduino IDE erleichtert Zugang zu Bibliotheken

und Tools.

Alle genannten Spezifikationsdaten entstammen dem ESP32-WROOM-32 Datenblatt [11].

Die Kombination aus all seinen Features macht den ESP32 zu einer perfekten Wahl für alle Entwickler die einen leistungsstarken und flexiblen Mikrocontroller mit guter Unterstützung suchen.

Rolle im Projekt

Im Rahmen dieser Arbeit ist der ESP32-WROOM-32 entscheidend für die Implementierung des sensorseitigen Wasserzeichens. Die Verwendung des Digital-Analog-Wandlers ermöglicht dem ESP32 eine präzise Spannungssteuerung für den Betrieb des Thermistors. Ausgehend von einer Basisspannung V_b, die als Ausgangswert dient, variiert der ESP32 diese Spannung geringfügig, um eine Bitsequenz darzustellen. Für ein Bit mit dem Wert 0 gibt der DAC die Spannung V_b aus, während für ein Bit mit dem Wert 1 die Spannung V_{b+wm} nach folgender Formel ausgegeben wird:

$$V_{b+wm} = V_b + V_{wm}$$

, wobei V_{wm} die Wasserzeichenspannung darstellt.

Entwicklungsumgebung und Programmierung

Die Programmierung des ESP32-WROOM-32 erfolgte in der Arduino IDE Umgebung, die den Programmierprozess durch umfangreiche Bibliotheken und Tools stark vereinfacht. Die Umgebung unterstützt direkt die Umsetzung der für das Projekt notwendigen Funktionen, wie z.B. die sichere Datenkommunikation mit dem WebSocket Server. Ein detaillierter Einblick in die Arduino IDE Entwicklungsumgebung wird im Abschnitt 3.2.1 gegeben.

Fazit

Der ESP32-WROOM-32 hat sich bei der Implementierung des sensorseitigen Wasserzeichen-Authentifizierungssystems als unverzichtbare Komponente erwiesen. Dank seiner leistungsfähigen Prozessorarchitektur, der umfangreichen Anschlussmöglichkeiten und der Möglichkeit, die Spannung über den Digital-Analog-Wandler präzise zu steuern, stellt er eine perfekte Lösung für die Arbeit dar. Besonders hervorzuheben ist die Verwendung der Arduino IDE für die Entwicklung, die den Programmierprozess durch den direkten Zugriff auf eine breite Palette von Bibliotheken und Tools erheblich vereinfacht hat.

3.1.2 NTC Thermistor 10K Ohm

Ein NTC-Thermistor (Negative Temperature Coefficient Thermistor) ist ein elektronisches Bauelement, dessen Widerstand mit steigender Temperatur abnimmt. Der verwendete Thermistor hat einen Nennwiderstand von 10 kOhm bei einer Bezugstemperatur von 25°C. Bei Temperaturen über 25°C nimmt der Widerstand exponentiell ab. Dieses Verhalten des NTC-Thermistors macht ihn zu einem effektiven Temperatursensor in einer Vielzahl von Anwendungen. Zum Beispiel zur Temperaturüberwachung und -regelung. Die hohe Empfindlichkeit und schnelle Reaktionszeit machen dieses elektronische Bauteil zu einer guten Wahl.[12]

Rolle im Projekt

Der 10KOhm NTC-Thermistor dient in unserem Projekt als Beispielobjekt zur Veranschaulichung der Einsatzmöglichkeiten des sensorseitigen Wasserzeichen-Authentifizierungssystems. Es ist wichtig zu betonen, dass die Temperaturänderung selbst keine direkte Rolle im Authentifizierungsprozess spielt. Vielmehr ist die durch den Digital-Analog-Wandler (DAC) des ESP32 hervorgerufene Spannungsmodulation entscheidend, wobei es wichtig ist, dass diese Modulation auch nach dem Durchlaufen des Thermistors noch erkennbar ist.

Aufgrund der fehlenden Relevanz der Temperaturänderung für die Funktionalität des Arbeitskonzeptes wird die Berechnung der Temperatur des ADC-Auslesewertes im weiteren Verlauf der Arbeit nur näherungsweise berechnet.

Spannungsteiler

Das Authentifizierungssystem dieser Arbeit verwendet Spannungsmodulation über eine vom DAC des ESP32 erzeugte Spannung, die verifizierbare Bitsequenzen im Spannungsfluss darstellen kann. Die Modulation und die anschließende Detektion sind zentral für das Konzept. Zur Messung der Thermistortemperatur und zur Regelung der am Spannungsteiler gemessenen Spannung V_S wird ein Spannungsteiler verwendet.

Ein gut abgestimmter Spannungsteiler ermöglicht es, die Ausgangsspannung so anzupassen, dass sie vom ADC optimal verarbeitet werden kann. Eine höhere Auflösung bedeutet, dass feinere Unterschiede in der modulierten Spannung beobachtet werden können, was die Genauigkeit des Authentifizierungsprozesses erhöhen kann. [13], [14]

Die Implementierung des Spannungsteilers im Konzept des sensorseitigen Wasserzeichen-Authentifizierungssystems

ermöglicht die Berechnung der Temperatur T über den Thermistorwiderstand R_T mit Hilfe der Beta-Gleichung, wie in der Arbeit [15, S. 4] beschrieben. Die Beta-Gleichung arbeitet, indem bei einer Temperatur T der Widerstand des Thermistors R_T in Beziehung zum Referenzwiderstand R_0 gesetzt wird, der den Widerstand des Thermistors bei der Referenztemperatur T_0 angibt. Durch die genaue Bestimmung von R_T aus der über dem Thermistor abfallenden Spannung des Spannungsteilers kann die aktuelle Temperatur aus der Beta-Gleichung berechnet werden.

Fazit

Zusammenfassend kann gesagt werden, dass der NTC-Thermistor in dieser Arbeit als Instrument zur Demonstration der Anwendbarkeit von Sensoren im Rahmen des Authentifizierungskonzepts verwendet wurde. Die entscheidende Komponente für das Konzept bleibt jedoch die Spannungsmodulation, die durch den DAC des ESP32 erzeugt wird. Diese Methodik unterstreicht die Vielseitigkeit und Anpassungsfähigkeit des Systems, das darauf ausgelegt ist, die Authentizität durch die Erkennung und Interpretation von Spannungssignalen unabhängig von der Art des verwendeten Sensors zu gewährleisten.

3.1.3 Digitales Potentiometer MCP-4131-104E/P

Das digitale Potentiometer MCP-4131-104E/P ist ein elektronischer Bautel, welcher die Funktion eines herkönnlichen Potentiometers digitalisiert. Er ermöglicht es über elektronische Signale den Widerstand dynamisch zu verändern. Der MCP hat einen Widerstandbereich von 0 bis 100k Ohm und bietet mit einer Auflösung von 7 bit eine präzise Einstellungsmöglichkeit. Die Steuerung erfolgt über eine serielle Schnittstelle namens SPI (Serial Peripheral Interface).[16]

MCP-4131-104E/P Spezifikationen:

- Widerstand R_{AB} von 100k Ohm
- 7-bit Auflösung: 128 Widerstände (129 Schritte)
- Minimale Betriebstemperatur: - 40°C
- Maximale Betriebstemperatur: + 125°C
- Digitale Schnittstelle: Serial (SPI)

Alle genannten Spezifikationsdaten entstammen dem MCP413X/415X/423X/425X Datenblatt [16].

Rolle im Projekt

Das digitale Potentiometer MCP-4131-104E/P ist ein wichtiges Element des sensorseitigen Wasserzeichen-Authentifizierungssystems und deckt 2 Funktionalitäten ab. Im Normalbetrieb kann das Bauteil zur Spannungsmodulation des Spannungsteilers verwendet werden. Zusätzlich spielt das Potentiometer eine wesentliche Rolle bei der Simulation von Angriffsszenarien.

Durch die dynamische Änderung des Widerstandswertes im Betrieb bis nahe 0 Ohm kann ein Zustand, bei dem das Ausgangssignal des Spannungsteilers durch einen externen Eingriff auf Masse gezogen wird, wirkungsvoll simuliert werden. Diese Möglichkeit, eine Grounding Attack ohne weitere Komponenten oder Eingriffe zu simulieren, erleichtert die funktionale Verifikation des Gesamtkonzeptes erheblich.

Fazit

Durch die Verwendung des digitalen Potentiometers MCP-4131-104E/P als Mittel zur Feinabstimmung der Spannung V_S am Spannungsteiler und zur Simulation von Angriffsszenarien wird nicht nur die Genauigkeit und Zuverlässigkeit des Geräts verbessert, sondern auch ein wesentlicher Beitrag zur Identifizierung potenzieller Schwachstellen geleistet.

3.1.4 Raspberry Pi 4

Der Raspberry Pi 4 gehört zur Serie der Raspberry Pi Mikrocomputer, die sich durch beeindruckende technische Eigenschaften und vielseitige Einsatzmöglichkeiten auszeichnen.

Raspberry Pi 4 Spezifikationen:

- Prozessor: Quad-Core ARM Cortex-A72 CPU mit bis zu 1,5 GHz.
- Speicher: 8 GB LPDDR4-2400 SDRAM.
- Konnektivität: Dual-Band 802.11 b/g/n/ac Wireless LAN, Bluetooth 5.0 mit BLE, Gigabit Ethernet.
- Videoausgang: Unterstützt Dual-Display-Ausgabe mit bis zu 4K-Auflösung über zwei Micro-HDMI-Anschlüsse.
- GPIO: 40-poliger GPIO-Header für Erweiterungen und Hardware-Interaktionen.

Alle genannten Spezifikationsdaten stammen aus dem Raspberry Pi 4 Datenblatt [17].

Die Kombination aus Leistung, Konnektivität und Erweiterbarkeit macht den Raspberry Pi 4 zu einem idea-

len Gerät für eine Vielzahl von Projekten, von einfachen DIY-Aufgaben bis hin zu komplexen, datenintensiven Anwendungen.

Rolle im Projekt

Im Rahmen dieser Arbeit dient der Raspberry Pi 4 als Gateway und zentrale Verarbeitungseinheit. Neben seiner Rolle als HAS Gateway übernimmt der Raspberry PI folgende zusätzliche Funktionen:

- WebSocket Server: Auf dem Raspberry Pi läuft ein WebSocket Server, der die Kommunikation zwischen dem Raspberry Pi und den WebSocket Clients, in unserem Fall dem ESP32, abwickelt.
- Webseite: Der Raspberry Pi stellt eine Webseite im lokalen Netzwerk zur Verfügung, über die der Benutzer das System überwachen kann. Es werden die aktuellen Sensorwerte und die Ergebnisse der Authentifizierungsprüfung angezeigt.
- Datenbank: Eine Datenbank mit zwei Datenbanktabellen dient zur Speicherung von Bitfolgen, Authentifizierungsergebnissen und Sensorwerten. Die Datenbank ist für den Authentifizierungsprozess sowie für die reibungslose Kommunikation mit den Clients unerlässlich.

Die Wahl des Raspberry Pi 4 für den Einsatz als HAS Gateway fiel aufgrund der Leistungsfähigkeit, der Funktionalität und der Flexibilität.

Fazit

Der Raspberry Pi 4 ist als leistungsstarke, flexible und gut unterstützte Plattform das Herzstück des sensorbasierten Wasserzeichen-Authentifizierungssystems. Seine Rolle als HAS-Gateway ermöglicht die effiziente Erfassung, Verarbeitung und Weiterleitung von Authentifizierungsdaten, während seine technischen Eigenschaften und die starke Unterstützung durch die Community die Entwicklung und Implementierung des Systems erleichtern.

3.2 Verwendete Software

Dieses Unterkapitel beschreibt die Softwareumgebung, die für die Entwicklung des sensorseitigen Wasserzeichen-Authentifizierungssystems verwendet wurde.

3.2.1 Arduino IDE

Die Arduino IDE in der Version 2.3.2 ist die Software-Entwicklungsumgebung, die für den Sensor-Client des sensorseitigen Wasserzeichen-Authentifizierungssystems verwendet wird. Die Arduino IDE basiert auf der Programmiersprache C/C++. Dadurch ist die Syntax sehr ähnlich und für Personen mit C und C++ Kenntnissen sehr einfach zu erlernen. Die Arduino IDE bietet eine Vielzahl von Werkzeugen, die das Programmieren und Debuggen stark vereinfachen. Eines dieser Tools ist der Library Manager, der es ermöglicht neben den offiziell vorinstallierten Arduino Libraries auch öffentliche und selbst erstellte Libraries zu verwenden. [18]

Die folgenden Bibliotheken wurden für die Implementierung des Clients verwendet:

- WiFi.h: Eine offizielle Arduino-Bibliothek, die den Anschluss an ein WiFi-Netzwerk ermöglicht. Sie bietet Funktionen zum Einrichten und Verwalten von WiFi-Verbindungen. Eine detaillierte Beschreibung aller Funktionen und Klassen, die mit dieser Bibliothek geliefert werden, ist in der offiziellen Arduino Dokumentation [18] zu finden.
- WebSocketsClient.h: Eine öffentliche Bibliothek, die die Verwendung von WebSockets unterstützt. Sie ermöglicht die bidirektionale Kommunikation zwischen Client und Server in Echtzeit. Eine detaillierte Beschreibung aller Funktionen und Klassen, die mit dieser Bibliothek ausgeliefert werden, ist im arduinoWebSockets GitHub Repository von Links2004 [19] zu finden.
- ArduinoJson.h: Eine öffentliche Bibliothek, die eine einfache Handhabung des JSON-Formats ermöglicht. Sie wird zum Generieren und Parsen von Dateien im JSON-Format verwendet, das für die WebSocket-Kommunikation verwendet wird. Eine detaillierte Beschreibung aller Funktionen und Klassen, die mit dieser Bibliothek ausgeliefert werden, ist auf der Website arduinojson.org des Entwicklers Benoît Blanchon [20] zu finden.
- SPI.h: Eine offizielle Arduino Library, die die SPI-Kommunikation zwischen dem zu programmierenden Gerät und anderen SPI-Geräten unterstützt. Eine detaillierte Beschreibung aller Funktionen und Klassen, die mit dieser Bibliothek geliefert werden, ist in der offiziellen Arduino Dokumentation [18] zu finden.

3.2.2 Raspberry Pi OS

Für den Raspberry Pi 4 wurde das mittlerweile zum Standard gewordene Betriebssystem Raspberry Pi OS verwendet. Das Betriebssystem basiert auf Debian und wurde speziell für die Raspberry Pi-Reihe entwi-

ckelt. Es zeichnet sich durch die Unterstützung des kompletten Raspberry Pi Toolsets und die detaillierte Beschreibung und Anleitung der Features aus, die in der offiziellen Raspberry Pi Dokumentation zu finden sind [21].

Als Programmiersprache für den auf dem Raspberry Pi 4 ausgeführten Code wurde Python gewählt, welches auf dem System bereits vorinstalliert ist. Wichtig hierbei ist, dass im Vergleich zu älteren Versionen des Betriebssystems keine Bibliotheken mehr direkt über den Paketinstaller pip installiert werden können. In den neuen Versionen ist es notwendig, eine virtuelle Umgebung zu erstellen, in der dann wie gewohnt Bibliotheken über den pip-Installer installiert werden können. [21]

Die folgenden Bibliotheken wurden für die Implementierung des HAS-Gateways des sensorseitigen Wasserzeichen-Authentifizierungssystems verwendet:

- asyncio: Eine Standardbibliothek in Python 3 für das Schreiben von asynchronem Code. Dokumentation: [22]
- aiohttp: Eine asynchrone HTTP-Client/Server-Framework für asyncio. Installation erforderlich: `pip install aiohttp`. Dokumentation: [23]
- websockets: Eine Bibliothek zur Erstellung von WebSocket Servern und Clients in Python mit asyncio. Installation erforderlich: `pip install websockets`. Dokumentation: [24]
- numpy: Eine Bibliothek, die Unterstützung für mehrdimensionale Arrays, Matrizen und einer Vielzahl von mathematischen Operation bietet. Installation erforderlich: `pip install numpy`. Dokumentation: [25]
- json: Eine Standardbibliothek in Python für das Parsen und Ausgeben von JSON-Daten. Dokumentation: [26]
- sqlalchemy: Eine SQL-Toolkit und Object-Relational Mapping (ORM) Bibliothek für Python. Installation erforderlich: `pip install SQLAlchemy`. Dokumentation: [27]

3.3 Testen der Hardware

3.3.1 Überprüfung DAC/ADC

Um die Funktionsfähigkeit des verwendeten ESP32 zu kontrollieren und um die Genauigkeit der DAC und ADC Komponente zu eruieren, wurde ein simpler Test durchgeführt. Der Test wurde mithilfe einer direkten Verbindung zwischen DAC und ADC, wie in Abb. 3.1 zu sehen ist, realisiert. Die direkte Verbindung er-

möglicht es eine Kennlinie zu erstellen, die etwaige Abweichungen der Komponenten aufzeigt. Des weiteren wurde die Korrelation zwischen dem DAC Ausgabewert und der tatsächlichen Spannung mittels Multimeter aufgezeichnet.

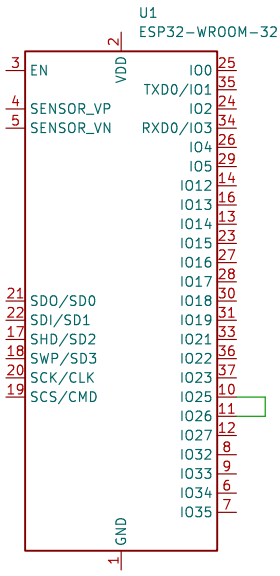

Abbildung 3.1: Simpler DAC/ADC Testaufbau

Zunächst wurde ein ESP32 Programm in der Arduino IDE erstellt, um die benötigten Kennwerte aufzeichnen zu können. Dabei wurden folgende Pins für die Komponenten DAC und ADC verwendet:

```
1 #define DAC 25
2 #define ADC 26
```

Listing 1: Code Snippet: Pinbelegung DAC/ADC

Im Hauptteil des Programms 2 werden verschiedene DAC-Werte ausgegeben und per ADC wieder eingelesen. Es wurden ADC-Aufzeichnungen für alle DAC-Werte von 0 - 255 gemacht. Zusätzlich wurden für die DAC-Ausgabewerte: 0, 32, 64, 96, 128, 160, 192, 224 und 255 bit manuelle Aufzeichnungen mittels Multimeter durchgeführt. Nach der Ausgabe der einzelnen DAC-Werte mittels dacWrite wird mittels des delay(1000)-Befehls eine Pause von 1 Sekunde eingefügt, um die entsprechenden Spannungswerte mittels Multimeter messen und aufzuzeichnen zu können. Nach der Pause wird, dann jeweils das Signal am ADC

eingelesen und über die Console ausgegeben.

```
#define DAC 25
#define ADC 26

int outputDAC = 0;

void setup() {
        Serial.begin(115200);
}

void loop() {
        if (outputDAC <= 255) {
                dacWrite(DAC, Voutput);
                delay(1000);
                Serial.println(analogRead(ADC));
                outputDAC++;
        }
}
```

Listing 2: Code: DAC/ADC

Der Test ermöglichte es, Datenpaare zwischen den DAC-Ausgangswerten und den entsprechenden ADC-Messwerten sowie den Multimeter-Messwerten zu sammeln. Das Ergebnis, zu sehen in 3.2, zeigt zunächst, dass die vom DAC ausgegebene Spannung nicht mit dem erwarteten Wert übereinstimmt (zu sehen in den Kennlinien Multimeter und Expected). Bei maximaler Ausgabe des 8-Bit-DACs wurde eine Spannung von 3,175V gemessen, was mehr als 0,12V unter der im Datenblatt [11] angegebenen Leistung liegt.

An der ADC-Kennlinie können zwei Phänomene beobachtet werden. Das erste Phänomen zeigt das Grundrauschen (Noise) des ADCs, welches durch die zitternden Abweichungen der Kennlinie sichtbar wird. Eine genauere Beschreibung des Rauschens des ADCs findet sich in der ESP32-Dokumentation von uPesy [28].

Das zweite Phänomen ist die Nichtlinearität des ADCs des ESP32, die in der Kennlinie an der starken Abweichung zu erkennen ist. Besonders auffällig ist auch, dass der ADC bei den ersten 3 Aufzeichnungen einen Wert von 0 und bei den letzten 5 Aufzeichnungen den Maximalwert von 4095 liefert. Die gemessenen Ergebnisse aus der Dokumentation von uPesy [28] und der Arbeit von [29, S. 4], sowie deren Aussagen, dass der ESP32-ADC Werte unter 0,1-0,2V nicht von 0V unterscheiden kann, stimmen mit den gemessenen Werten gut überein.

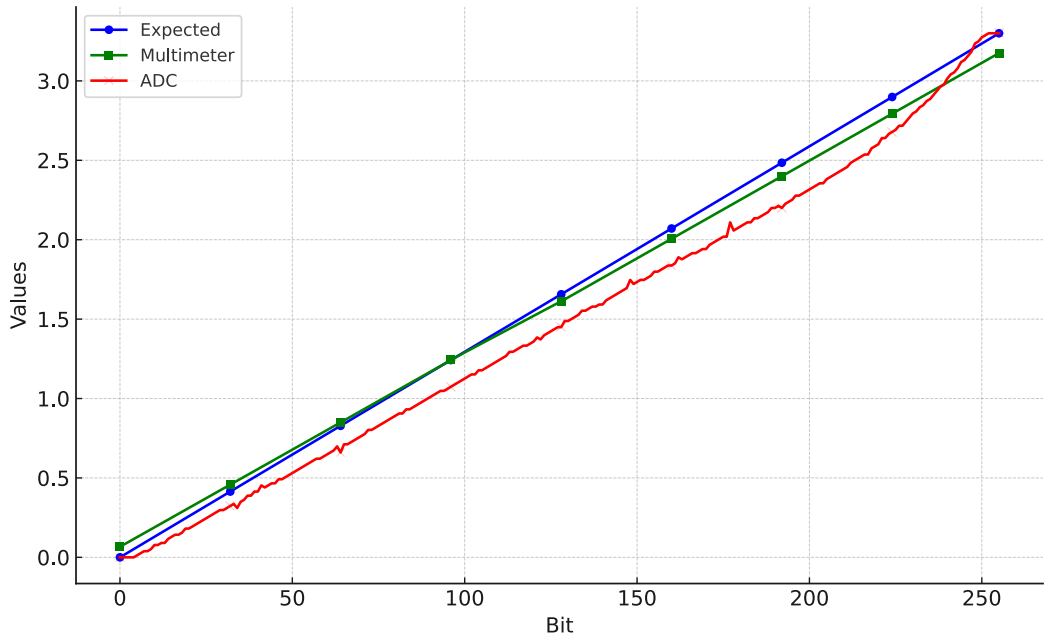

Abbildung 3.2: Abweichung DAC/ADC/Multimeter

Die durch den Test gewonnenen Einblicke über die Genauigkeit der DAC- und ADC-Komponenten des ESP32 bieten eine gute Ausgangslage für die Bewertung und Analyse der Ergebnisse des endgültigen Konzeptes.

3.3.2 Thermistor Funktionalität

Um die Funktionalität des NTC-Thermistors zu überprüfen, wurde eine Testschaltung aufgebaut und ein Test durchgeführt. Der Test beinhaltet die Spannungsversorgung des Thermistors über den DAC und das Auslesen des ADC, um den erhaltenen Wert in eine Temperatur umzuwandeln. Für den Testaufbau wurde der NTC-Thermistor mit einem 10K Ohm Widerstand in einen Spannungsteiler zwischen DAC und ADC

geschaltet, wie in Abbildung 3.3 zu sehen ist. Wie in Abschnitt 3.1.2 beschrieben, reagieren Thermistoren nichtlinear auf Temperaturänderungen, d.h. der Widerstand des Bauteils nimmt nicht linear mit der Temperatur ab.

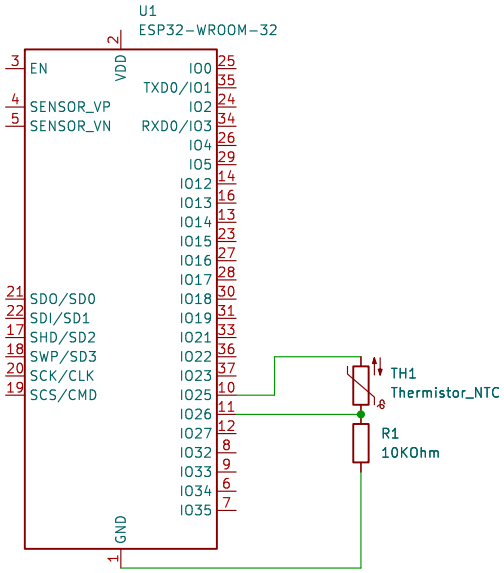

Abbildung 3.3: Thermistor Test Aufbau

Um den Widerstand eines Thermistors einfach und genau berechnen zu können, wird, wie im White Paper *An Explanation of the Beta and Steinhart-Hart Equations for Representing the Resistance vs. Temperature Relationship in NTC Thermistor Materials* von QTI Sensing Solutions Engineering Department [15, S. 3] beschrieben, die Beta-Gleichung verwendet, welche die Temperatur T wie folgt berechnet:

$$T = \frac{1}{\frac{1}{T_0} + \frac{1}{\beta} \ln\left(\frac{R_T}{R_0}\right)}$$

Dabei ist:

- T_0 die Referenztemperatur in Kelvin (25°C sprich 298,15 K),
- β der Beta-Wert des Thermistors,
- R_T der Widerstand des Thermistors bei der Temperatur T,
- R_0 der Widerstand des Thermistors bei der Referenztemperatur T_0,
- $\ln$ der natürliche Logarithmus.

Da sowohl die Versorgungsspannung V_{max} als auch die Referenzspannung des ADC V_{ref} 3,3 V betragen, kann diese Formel, wie im Artikel *Temperature Measurement with NTC Thermistors* von Kane [30] beschrieben, wie folgt vereinfacht werden:

$$T = \frac{1}{\frac{1}{T_0} + \frac{1}{\beta} \ln\left(\frac{adcMaxValue}{adcRawVal}\right) - 1}$$

Dabei ist:

- `adcMaxValue` der maximale am ADC einlesbare Wert in Bit,
- `adcRawVal` der am ADC eingelesene Wert in Bit.

Im Codeausschnitt Listing 3 werden die wichtigsten Variablen der Beta-Gleichung deklariert, wobei der Wert β durch den Variablennamen thermistorBetaValue und der Wert T_0 durch den Variablennamen referenceTemperatureKelvin ersetzt wurde. Die Variablen temperatureKelvin und temperatureCelsius werden für die zu berechnende Temperatur in der jeweiligen Temperaturskala verwendet.

```
1  #define DAC 25
2  #define ADC 26
3  // Konstanten
4  const double adcMaxValue = 4095.0;
5  const double thermistorBetaValue = 4050.0;
6  const double referenceTemperatureKelvin = 298.15;
7  // Variablendeklarationen
8  double adcRawValue;
9  double temperatureKelvin;
10 double temperatureCelsius;
11
12 void setup() {
13 Serial.begin(115200);
14 }
```

Listing 3: Codeausschnitt: Thermistor Funktionalität - Definitionen, Variablen und Setup

Die Testschleife in Listing 4 ist für die Temperaturmessung und -berechnung zuständig. Das Programm durchläuft folgende Schritte:

1. Zuerst wird mit `dacWrite(DAC, 255);` die maximale DAC-Ausgangsspannung von 3,3V an den Thermistor ausgegeben.

2. Der aktuelle Spannungswert am ADC-Pin, der den Spannungsteiler ausliest, wird mit `adcRawValue = analogRead(ADC);` erfasst. Dieser Wert wird für die Berechnung der Temperatur benötigt.
3. Die Temperatur in Kelvin wird mit der oben definierten vereinfachten Beta-Gleichung berechnet: `temperatureKelvin = 1 / ((1/referenceTemperatureKelvin) + (1/thermistorBetaVal * log((adcMaxValue / adcRawValue) - 1));`.
4. Anschließend wird die Temperatur von Kelvin in Celsius umgerechnet: `temperatureCelsius = temperatureKelvin - 273.15;`.
5. Die Anweisung `delay(2000);` am Ende sorgt für eine Pause von 2 Sekunden zwischen den Messungen. Dies hilft, den korrekten Ablauf des Tests mit manuellen Messungen zu kontrollieren.

```
1 void loop() {
2   dacWrite(DAC, 255);
3   adcRawValue = analogRead(ADC);
4   temperatureKelvin = 1 / ((1/referenceTemperatureKelvin) +
    ↪  (1/thermistorBetaValue) * log((adcMaxValue / adcRawValue) -
    ↪  1));
5   temperatureCelsius = temperatureKelvin - 273.15;
6   delay(2000);
7 }
```

Listing 4: Codeausschnitt: Thermistor Funktionalität - Loop Funktion

Die Abb. 3.4 zeigt das Ergebnis des Thermistortests über einen Zeitraum von 288 Sekunden. Die Linie zeigt deutlich, dass selbst bei konstanter Versorgungsspannung der am ADC ausgelesene Wert nicht konstant bleibt. Dies könnte zum einen an der im Abschnitt 3.3.1 dargestellten Nichtlinearität und dem Grundrauschen der elektronischen Bauteile liegen, zum anderen aber auch am Thermistor selbst. Genauere Analysen und die Auswirkung am Konzept werden im Kapitel 5 und Kapitel 6 durchgeführt.

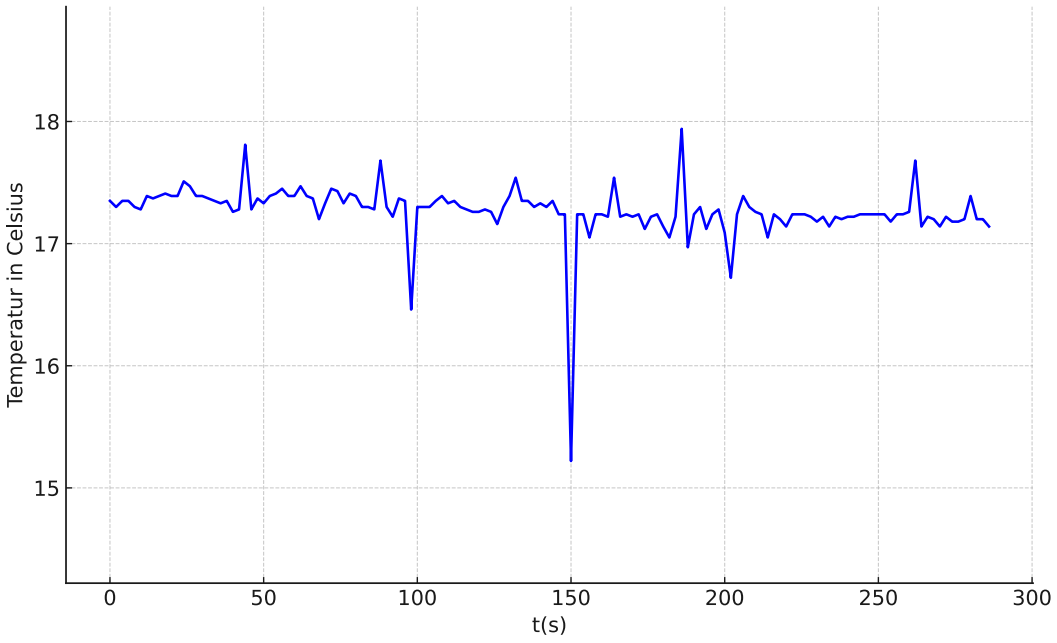

Abbildung 3.4: Temperaturverlauf Thermistor

3.3.3 WLAN Chip

Um die WLAN Funktionalität zu testen wurde ein Beispielprogramm Connecting to a Wi-Fi Access Point der ESP32 Dokumentation von uPesy, mittels WiFi.h-Library, verwendet. [28]

Der Code im Listing 5 aus der ESP32-Dokumentation [28] demonstriert die Vorgehensweise zum Aufbau einer WiFi-Verbindung auf einem ESP32. Die wichtigsten Schritte sind wie folgt strukturiert:

1. Durch `include <WiFi.h>` am Anfang des Codes wird die in Abschnitt 3.2.1 beschriebene Bibliothek `WiFi.h` eingebunden, die die grundlegende Funktionalität für die Netzwerkverbindung bereitstellt.
2. Die Variablen `ssid` und `password` werden mit den Zugangsdaten des zu verbindenden WiFi-Netzwerks initialisiert. Diese Informationen wurden im Listing mit Dummy-Werten versehen.
3. In der `setup()` Funktion wird zunächst die serielle Kommunikation mit `Serial.begin(115200);` initialisiert und eine kurze Verzögerung eingefügt, um sicherzustellen, dass der Serial Monitor rechtzeitig geöffnet werden kann.
4. Der WiFi-Modus wird mit `WiFi.mode(WIFI_STA);` eingestellt. `WIFI_STA` steht für den Sta-

```
#include <WiFi.h>

const char* ssid = "yourNetworkName";
const char* password = "yourNetworkPassword";

void setup(){
        Serial.begin(115200);
        delay(1000);

        WiFi.mode(WIFI_STA); //Optional
        WiFi.begin(ssid, password);
        Serial.println("\nConnecting");

        while(WiFi.status() != WL_CONNECTED){
                Serial.print(".");
                delay(100);
        }

        Serial.println("\nConnected to the WiFi network");
        Serial.print("Local ESP32 IP: ");
        Serial.println(WiFi.localIP());
}

void loop(){}
```

Listing 5: Code: Test WiFi [28]

tionsmodus. Der Modus wird verwendet, um anzugeben, dass sich der ESP32 als Client mit einem Access Point verbindet.

5. Der Verbindungsaufbau zum Netzwerk wird mit `WiFi.begin(ssid, password);` eingeleitet. Dabei werden die zuvor definierten Variablen `ssid` und `password` verwendet.
6. Eine `while`-Schleife prüft kontinuierlich den Verbindungsstatus mit `WiFi.status()`. Die Schleife läuft solange, bis der Status `WL_CONNECTED` erreicht wird. Dieser Status zeigt an, dass der Verbindungsaufbau erfolgreich war.
7. Nach erfolgreichem Verbindungsaufbau werden Informationen zur Verbindung über die serielle Schnittstelle ausgegeben, einschließlich der lokalen IP-Adresse des ESP32, was die erfolgreiche Verbindung zusätzlich bestätigt.

Bei erfolgreicher Verbindung sieht der Output im Serial Monitor wie folgt aus:

```
Connecting
.......
Connected to the WiFi network
Local ESP32 IP: 192.168.111.11
```

3.3.4 Potentionmeter Funktionalität

Um die Funktionalität des digitalen Potentiometers MCP4131-104E/P zu testen, wurde dieser an den ESP32 angeschlossen und ein Programm erstellt. Für die korrekte Verbindung wurde das digitale Potentiometer nach Angaben des Datenblattes [16], wie in Abbildung 3.5 zu sehen, mit dem ESP32 verbunden. Wichtig der PA0 Pin wurde im Zuge der Arbeit nicht verwendet, da nur einer der nicht Wiper Pins für das Konzept benötigt wird.

Nach erfolgreicher Umsetzung des Testaufbaus wurde ein Programm angefertigt, welches im restlichen Kapitel in 2 Teilen beschrieben wird. Der erste Teil des Programms 6 legt die Grundlagen für die Kommunikation mit dem digitalen Potentiometer über die SPI-Schnittstelle. Die wichtigsten Schritte umfassen:

1. Mit der Anweisung `include <SPI.h>` wird die im Abschnitt 3.2.1 beschriebene SPI-Bibliothek eingebunden. Die Bibliothek ermöglicht die Kommunikation über SPI.
2. Zuweisung der für die Kommunikation benötigten Pins:
    - Die Pins `DAC` und `ADC` werden wie in den vorherigen Tests für die Verwendung von DAC und ADC benötigt. Eine wichtige Änderung gegenüber den vorherigen Tests besteht darin, dass der

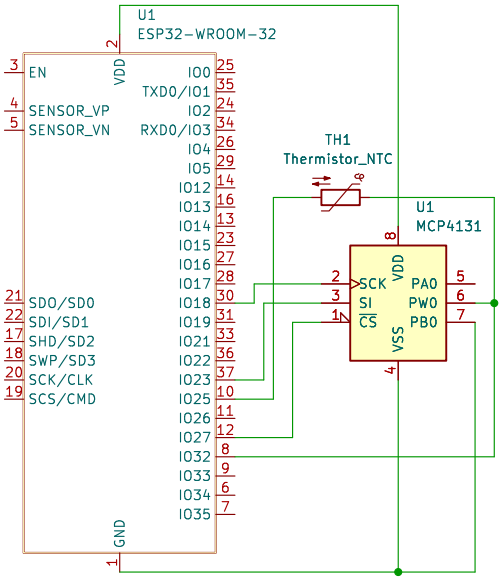

Abbildung 3.5: Potentiometer Test Aufbau

ADC-Pin auf Pin 32 gelegt wurde, da der ADC-Pin 26, obwohl für diesen Test nicht relevant, nicht in Verbindung mit der WiFi-Funktionalität des ESP32 verwendet werden kann. [28]

- `CS_PIN` definiert den Pin für das Chip-Select-Signal, das das digitale Potentiometer aktiviert und deaktiviert.
- `MOSI_PIN` und `SCK_PIN` definieren die Pins für das Master-Out-Slave-In-Signal (MOSI) und das Clock-Signal (SCK) der SPI-Kommunikation.

3. Die Funktion `setup()` initialisiert die serielle Kommunikation und konfiguriert das Chip-Select-Signal:
   - Die serielle Kommunikation des ESP32 wird mit einer Baudrate von 115200 gestartet.
   - Der Chip-Select-Pin wird als Output konfiguriert, da der ESP32 über diesen Pin das digitale Potentiometer für die Kommunikation vorbereitet.
   - Mit `SPI.begin()` wird die SPI-Kommunikation gestartet. Dieser Schritt ist notwendig, um die Datenübertragung zwischen dem Mikrocontroller und dem digitalen Potentiometer zu ermöglichen. Da das digitale Potentiometer keinen MISO-Pin besitzt, wird dieser nicht automatisch einem ESP32-Pin zugewiesen.
   - Schließlich wird die selbst erstelle Funktion `setDigitalPotentiometer` mit dem Wert 13 ausgeführt.

Wie in Abschnitt 3.1.3 beschrieben, ist das digitale Potentiometer MCP-4131-104E/P ein 100k Ohm Widerstand mit einer Auflösung von 7 Bit. Zur korrekten Einstellung des Potentiometers muss der gewünschte

Widerstandswert in eine Bitzahl von 0-128 umgewandelt werden. Der gesuchte Bitwert für einen bestimmten Widerstandswert kann mit der folgenden Formel berechnet werden:

$$MCP\_Wert = \left\lfloor \frac{R_{\text{gewünscht}}}{R_{\text{max}}} \times 128 \right\rfloor$$

Dabei ist:

- $R_{\text{gewünscht}}$ der Widerstandswert, für den eine Bitzahl ermittelt werden soll,
- R_{max} der maximale Widerstandswert des digitalen Potentiometers.

Um einen Widerstandswert von 10k Ohm einzustellen, muss dem Potentiometer der Wert 13, wie im Codeausschnitt 6, übermittelt werden.

```
#include <SPI.h>
#define DAC 25
#define ADC 32
const int CS_PIN = 27;
const int MOSI_PIN = 23;
const int SCK_PIN = 18;

void setup() {
        Serial.begin(115200);
        pinMode(CS_PIN, OUTPUT);
        SPI.begin(SCK_PIN, MISO, MOSI_PIN, CS_PIN);
        setDigitalPotentiometer(13);
}
```

Listing 6: Codeausschnitt: Deklaration und Einstellung des MCP-4131

Der zweite Teil des Programms besteht aus zwei Komponenten: der Definition der Funktion setDigitalPotentiometer, die es ermöglicht, den Widerstandswert des digitalen Potentiometers einzustellen, und der Funktion loop, die den für den Test erforderlichen Ablauf enthält. Die Schritte sind wie folgt strukturiert:

1. `setDigitalPotentiometer` ist eine benutzerdefinierte Funktion, die die notwendigen Schritte durchführt, um den Widerstand des Potentiometers auf einen übergebenen Wert $value$ zu ändern. Die Funktion besteht aus den folgenden Schritten:

- Starten der Kommunikation mit dem MCP-4131 durch das Setzen des Chip-Select-Pins (`CS_PIN`) auf `LOW`.
- Senden des Befehlsbytes (`B00000001`) an das Potentiometer, um den folgenden Wert als neuen Widerstandswert festzulegen.
- Senden des gewünschten Widerstandswertes (`value`) über die SPI-Schnittstelle.
- Beenden der Kommunikation mit dem Potentiometer durch Setzen von (`CS_PIN`) auf `HIGH`.

2. Die Funktion `loop` enthält und führt die folgenden Aktionen für den für den Test erforderlichen Ablauf aus:
   - Ausgabe der maximalen Spannung am DAC-Pin (`DAC`) mit `dacWrite`.
   - Auslesen des aktuellen Spannungswertes am ADC-Pin (`ADC`) und Ausgabe dieses Wertes über die serielle Schnittstelle.
   - Einstellen des digitalen Potentiometers auf seinen maximalen Widerstandswert (100k) mit dem Funktionsaufruf `setDigitalPotentiometer(127)`.
   - Pause von zwei Sekunden (`delay(2000)`), gefolgt von erneutem Einlesen und Ausgeben des ADC-Wertes.

```
1  void setDigitalPotentiometer(int value) {
2          digitalWrite(CS_PIN, LOW);
3          SPI.transfer(B00000001);
4          SPI.transfer(value);
5          digitalWrite(CS_PIN, HIGH);
6  }
7
8  void loop() {
9          dacWrite(DAC_CH1, 255);
10         Serial.println(analogRead(ADC));
11         setDigitalPotentiometer(128);
12         delay(2000);
13         Serial.println(analogRead(ADC));
14 }
```

Listing 7: Codeausschnitt: Loop des MCP-4131

Der durchgeführte Test ergab folgende Mittelwerte am ausgewählten ADC:

- 1730 bei einem Potentiometerwert von 13
- 3450 für einen Potentiometerwert von 128

3.4 Implementierung Wasserzeichen

Die Implementierung eines wirksamen Wasserzeichens für die Kommunikation innerhalb von Heimautomatisierungssystemen (HAS) stellt eine große Herausforderung dar, die eine sorgfältige Abwägung zwischen Erkennbarkeit und Vermeidung von Störungen der normalen Systemfunktionalität erfordert. Aufbauend auf den Erkenntnissen der Arbeit von Ruotsalainen et al. [1], verfolgt diese Diplomarbeit einen Ansatz, der auf der Einbettung eines subtilen, aber eindeutig identifizierbaren Signals in die Sensorversorgung basiert. Diese Signalmodulation, das wir als Wasserzeichen bezeichnen, ist so konzipiert, dass sie robust gegen alltägliche Störungen und eine Vielzahl von Angriffen ist, ohne dabei die Kerneigenschaften des Systems zu beeinträchtigen.

3.4.1 Sensor Bias

Ein zentrales Problem bei der Implementierung des sensorseitigen Wasserzeichens, wie in der Arbeit *Watermarking Based Sensor Attack Detection in Home Automation Systems* von Ruotsalainen et al. dargestellt, ist die Bias-Induktion, die durch das Hinzufügen einer Spannung V_{wm} zur Darstellung eines Wasserzeichenbits mit dem Wert 1 entsteht[1]. Diese Spannungserhöhung kann das Ergebnis des Sensors beeinflussen, was besonders in Systemen, die auf genauen Sensordaten basieren, problematisch ist.

Als Lösung für dieses Problem schlagen Ruotsalainen et al. die Anwendung des Manchester Code II vor und verwenden diesen auch. Der Manchester Code II ist ein Kodierungsverfahren, das jedes zu übertragende Bit in zwei Signale umwandelt, wobei ein '1'-Bit durch eine '1' gefolgt von einer '0' und ein '0'-Bit durch eine '0' gefolgt von einer '1' repräsentiert wird [1]. Diese Methode sorgt für eine ausgeglichene Spannungsmodulation, indem für jedes Bit des Wasserzeichens ein Bitpaar erzeugt wird, das aus aufeinander folgenden hohen und niedrigen Spannungen besteht. Der Bias wird neutralisiert, indem nur der Messwert des Sensors verwendet wird, der unter Verwendung des Bits des Manchester-Paares mit dem Wert 0 aufgezeichnet wird, da bei einem Bit mit dem Wert 0, wie oben beschrieben, keine Erhöhung der Spannung V_b mit V_{wm} erfolgt. Diese ausgewogene Modulation macht den Manchester-II-Code zu einer idealen Lösung für die Herausforderungen, die mit der Bias-Induktion durch das sensorseitige Wasserzeichen verbunden sind.

Manchester II Kodierung

Für die Kodierung der Wasserzeichen-Bitsequenz im Manchester-II-Format wird die benutzerdefinierte Funktion `manchesterEncode`, zusehen im Listing 8, auf dem ESP32 verwendet.

```
String manchesterEncode(const String &originalSequence) {
        String encodedSequence = "";
        for (char bit : originalSequence) {
                switch (bit) {
                        case '1':
                        encodedSequence += "10";  // Encode '1' as
                         ↪  "10"
                        break;
                        case '0':
                        encodedSequence += "01";  // Encode '0' as
                         ↪  "01"
                        break;
                        default:
                        Serial.println("Error: Invalid character in
                         ↪  sequence.");
                        break;
                }
        }
        return encodedSequence;
}
```

Listing 8: Code: Manchester II Kodierung

Die Funktion `manchesterEncode` durchläuft die folgenden Schritte:

1. Definition der Funktion `manchesterEncode`: Diese benutzerdefinierte Funktion kann mit einer Bitfolge vom Datentyp `String` aufgerufen werden und liefert die Manchester-kodierte Sequenz als

`String` zurück.

- Die Funktion initialisiert einen leeren `String`. `encodedSequence`, in dem die codierte Sequenz zwischengespeichert wird.
- Für jedes Zeichen (Bit) der an die Funktion übergebenen Sequenz wird die Schleife einmal durchlaufen.

2. `for (char bit : originalSequence)` Für jedes Bit in der übergebenen Sequenz wird eine Entscheidung in Abhängigkeit von seinem Wert mit der Switch-Anweisung `switch (bit)` getroffen.
   - Wenn das Bit '1' ist, wird der `String "10"` an `encodedSequence` angehängt, was der Manchester-Codierung von '1' entspricht.
   - Wenn das Bit '0' ist, wird `"01"` hinzugefügt, was der Manchester-Codierung von '0' entspricht.
   - Für jedes Zeichen, das nicht '0' oder '1' ist, gibt die Funktion eine Fehlermeldung über die serielle Schnittstelle aus.
3. `return encodedSequence;` gibt am Ende der Schleife die codierte Sequenz `encodedSequence` zurück, die nun vollständig Manchester-codiert ist.

3.4.2 Evaluierung der Wasserzeichen Parameter

Um das Potential unseres wasserzeichenbasierten Detektionssystems über einen weiten Parameterbereich zu evaluieren, wurden der Testaufbau 3.6 und der Testcode, welcher hier in die vier Abschnitte 9, 10, 11 und 12, unterteilt wurde, entwickelt. Dieser Ansatz folgt der Methodik von Ruotsalainen et al. [1, S. 5], und ermöglicht die Bestimmung der Bitfehlerrate (BER) durch verschiedene Kombinationen von V_{wm} und V_S. Konkret besteht das Testwasserzeichen aus einer zufällig generierten, gleich verteilten 256-Bit-Sequenz, die umfassend über alle möglichen Kombinationen getestet wird. Dabei konzentrieren wir uns auf den Bereich $0 \leq V_{\text{wm}} \leq 0.077\text{V}$ für V_{wm} und $0 \leq V_{\text{S}} \leq 1.5\text{V}$ für V_S, innerhalb dessen wir die BER für jede Einstellung bestimmen.

Der größe Unterschied zur Arbeit *Watermarking Based Sensor Attack Detection in Home Automation Systems* [1] ist, dass der dort verwendete Mikrocontroller *ST Microsystems development board B-L072Z-LRWAN1* eine wesentlich niedrigere Referenzspannung mit 1,224V für die DAC/ADC-Kombination hat als der ESP32 mit 3,3V. Dieser Unterschied hat enorme Auswirkungen, wenn bedacht wird, dass der DAC des ESP32 nur über 8 Bit anstelle von 12 Bit verfügt. Dieser qualitative Unterschied der Mikrocontroller hat zur Folge, dass der im Paper für den Parameter V_{wm} getestete Bereich von $0 \leq V_{\text{wm}} \leq 7.5\text{mV}$ für V_{wm} kleiner ist als die

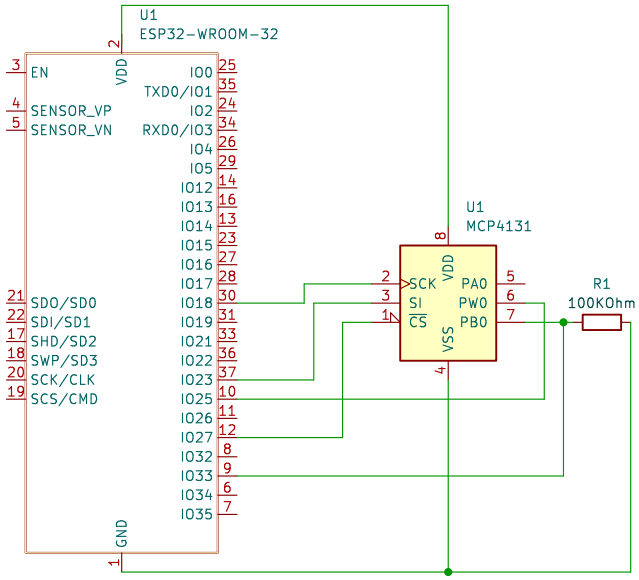

Abbildung 3.6: Testaufbau Evaluierung der Wasserzeichen Parameter

Spannungsdifferenz, die durch den kleinstmöglichen Schritt des DAC des ESP32 entsteht. Der kleinstmögliche Schritt mit nur einem Bit entspricht einer Spannung von 12,89mV, also 5mV mehr als der größte Wert, der für V_{wm} getestet wurde. Ein weiteres Problem des ESP32 ist die Nichtlinearität und das ausgeprägte Grundrauschen der DAC/ADC-Kombination, zwei Phänomene die in der Abbildung 3.1 beobachtet werden können.

Wie in der Abbildung 3.6 zu sehen wurde für den Testaufbau der NTC-Thermistor durch einen 100k Ohm Widerstand ersetzt. Der Vorteil dieses Testaufbaus ist, dass mit dem Widerstand die Effizienz des Systems unabhängig vom verwendeten Sensor evaluiert werden kann. Als weitere Änderung wurde die Positionierung von Potentiometer innerhalb des Spannungsteilers geändert. Durch die Verwendung des MCP-4131 als erster Widerstand des Spannungsteilers kann durch das Widerstandsverhältnis zum 100k Ohm Widerstand bei einer DAC Ausgangsspannung von 3V einfach durch den Spannungsbereich $0 \leq V_S \leq 1.5V$ für V_S durch iteriert werden.

Der folgende Codeausschnitt 9 führt die Initialisierung und das Setup für die Evaluierung durch, indem spezielle Hardwarekonfigurationen für die SPI-Kommunikation vorgenommen werden. Der Codeausschnitt ist in folgende Schritte untergliedert:

1. Der erste Teil des Codeausschnittes beinhaltet die Einbindung der SPI-Bibliothek sowie die Definition

der für die SPI-Kommunikation notwendigen Konstanten und Variablen. Eine genaue Erklärung der Variablen und Konstanten ist im Abschnitt 3.3.4 für den Code 6 zu finden.

2. Der zweite Teil des Codeausschnitts enthält die Setup-Funktion `setup()`, mit der die serielle Kommunikation initialisiert und das Chip-Select-Signal konfiguriert wird. Eine genaue Erläuterung der einzelnen Befehle findet sich wie beim ersten Teil des Codeausschnitts im Abschnitt 3.3.4 für den Code 6.

Zusammengefasst wird durch die oben beschriebenen Schritte das System für die Evaluierung vorbereitet. Diese Vorbereitung umfasst das Einbinden der notwendigen Bibliothek, die Definition wichtiger Konstanten und Variablen, und das Initialisieren der Kommunikationsprotokolle und Pin-Konfigurationen. Der Fokus liegt auf der Einrichtung der SPI-Kommunikation, was für den Datenaustausch mit SPI-fähigen Geräten wie in diesem Fall des digitalen Potentiometers notwendig ist.

```
1  #include <SPI.h>
2  #define DAC 25
3  #define ADC 33
4  const int CS_PIN = 27;
5  const int MOSI_PIN = 23;
6  const int SCK_PIN = 18;
7  void setup() {
8          Serial.begin(115200);
9          pinMode(CS_PIN, OUTPUT);
10         SPI.begin(SCK_PIN, MISO, MOSI_PIN, CS_PIN);
11 }
```

Listing 9: Codeausschnitt: Initialisierung und Setup der Evaluierung

Im Codeabschnitt 10 befinden sich benutzerdefinierte Funktionen, für die Steuereung des digitalen Potentiometer sowie für die Kodierung einer Bit-Sequenz mithilfe des Manchester II Codes.

1. Die erste benutzerdefinierte Funktion `setDigitalPotentiometer(int value)` dient dazu, den Wert eines digitalen Potentiometers über die eingerichtete SPI-Kommunikation einzustellen. Eine

genaue Erläuterung der einzelnen Befehle findet sich im Abschnitt 3.3.4 für den Code 7.

2. Die zweite benutzerdefinierte Funktion `manchesterEncode(const String &originalSequence)` wandelt eine gegebene Bitsequenz in eine Manchester-kodierte Sequenz um. Eine genaue Erläuterung der einzelnen Befehle findet sich im Abschnitt 3.4.1 für den Code 8.

```
void setDigitalPotentiometer(int value) {
        digitalWrite(CS_PIN, LOW);
        SPI.transfer(B00000001);
        SPI.transfer(value);
        digitalWrite(CS_PIN, HIGH);
}
String manchesterEncode(const String &originalSequence) {
        String encodedSequence = "";
        for (char bit : originalSequence) {
                switch (bit) {
                        case '1':
                        encodedSequence += "10";
                        break;
                        case '0':
                        encodedSequence += "01";
                        break;
                        default:
                        Serial.println("Error: Invalid char.");
                        break;
                }
        } return encodedSequence;
}
```

Listing 10: Codeausschnitt: Zusatzfunktionen der Evaluierung

Der Codeausschnitt 11 zeigt den ersten Teil der Evaluierungslogik welcher mithilfe der Funktion `loop()` die zuvor definierten Bereiche für V_S und V_{wm} durchläuft. Zusätzlich werden die für die Logik benötigten Variablen initialisiert.

1. Definition von Basiskonstanten und Sequenzen:
   - `const int baselineDacValue = 232;` definiert den Baseline-Wert des DAC für eine Ausgangsspannung von etwa 3V.
   - `const int remainingSteps = 6;` legt die Anzahl der DAC-Schritte fest die für den definierten Testbereich von V_{wm} benötigt werden. Ein DAC Wert von 6 entspricht 0,077V.
   - `String originalSequence = "...";` enthält die Bitfolge, die als Beispiel für die Evaluierung dient. Die Sequenz besteht aus 256 gleichmäßig verteilten Bits, die zufällig gemischt wurden.
   - `String encodedSequence;` wird verwendet, um die Manchester-kodierte Version der Bit-Sequenz zu speichern.
   - `char bitValidation = 0;` und `int stepSize = 128 / 15;` sind vorbereitende Variablen für die Evaluierungslogik, wobei `stepSize` die Größe eines jeden Potentiometer-Schritts berechnet. Für den Test wurde durch den definierte Bereich von V_S in 15 Schritten durch iteriert.
2. Die Funktion `loop()` enthält die Logik für die Iteration der zu testenden Bereiche. In der Funktion werden folgende Schritte ausgeführt
   - Zunächst wird die Originalsequenz mit der `manchesterEncode` Funktion10 codiert und in der Variable `encodedSequence` gespeichert.
   - Eine `for`-Schleife (`for (int potiValue = 0; potiValue <= 129; potiValue += 17)`) iteriert über die Potentiometerwerte (`potiValue`), die dem digitalen Potentiometer von 0 bis 129 in 8er-Schritten übergeben werden, um den Wert von V_S schrittweise zu ändern.
   - Die Funktion `setDigitalPotentiometer(potiValue);` 10 wird in jedem Schritt der Schleife aufgerufen, um den Widerstand des digitalen Potentiometers auf den aktuellen Wert von `potiValue` zu stellen.
   - Eine zweite `for`-Schleife (`for (int step = 0; step <= 6; step ++)`) iteriert über die DAC-Werte, die dem DAC von 0 bis 6 übergeben werden, um den Wert V_{wm} zu ändern.
   - Der Funktion `transmitSequence(potiValue, step);` werden bei jedem Durchlauf die aktuellen Schrittweiten übergeben.

- Die `while (true) delay(1000);` Schleife sorgt dafür, dass das Programm am Ende des Tests in einer Endlosschleife hängen bleibt.

```
1  const int baselineDacValue = 232;
2  const int remainingSteps = 6;
3  String originalSequence = ""; // 256-bit sequence
4  String encodedSequence;
5  char bitValidation = 0;
6  int stepSize = 128 / 15;
7  void loop() {
8          String encodedSequence =
           ↪  manchesterEncode(originalSequence);
9          for (int potiValue = 0; potiValue <= 129; potiValue += 17)
           ↪  {
10                 setDigitalPotentiometer(potiValue);
11                 for (int step = 0; step <= 6; step ++) {
12                         transmitSequence(potiValue, step);
13                 }
14         }
15         while (true) delay(1000);
16 }
```

Listing 11: Codeausschnitt: Evaluierungs loop + Variablen

Im letzten Abschnitt 12 wird die Implementierung des Wasserzeichens innerhalb des DAC-Signals umgesetzt und für diesen Test auch authentifiziert. Für die Implementierung wird die Funktion `transmitSequence (int potiValue, int Vwm)` verwendet, die innerhalb der Schleife mit den Parametern `int potiValue` und `int Vwm` aufgerufen wird. In der Funktion werden die folgenden Schritte zur Realisierung des sensorseitigen Wasserzeichens durchgeführt:

1. Die Variable `int errors = 0;` initialisiert einen Zähler zur Erfassung der falsch erkannten Bits der Sequenz. Der Zähler wird in weiteren Schritten zur Berechnung der BER benötigt.

2. Durchlauf der Manchester-kodierten Sequenz: Die angegebene `for`-Schleife `for (int i = 0; i < encodedSequence.length(); i += 2)` iteriert über jedes Bitpaar der codierten Sequenz. Die folgenden Aktionen werden für jedes Bitpaar ausgeführt:
   - Berechnung der DAC-Werte und Auslesen der ADC-Werte: Für jeden Teil des Manchester-Bitpaares wird die für das Wasserzeichen benötigte Ausgangsspannung berechnet, indem der Basiswert des DAC-Wertes (`baselineDacValue`) angepasst wird. Im Vergleich zur normalen Wasserzeichen-Implementierung im Abschnitt 3.6 wird hier, um die Spannung V_S für den Test nicht zu erhöhen bei einem Bit mit Wert '1' die Ausgangsspannung V_b und nicht V_{b+wm} berechnet. Stattdessen wird für ein Bit mit dem Wert '0' die Spannung V_{b-wm} berechnet. Die berechnete Spannung wird dann mit `dacWrite` am DAC ausgegeben. Nach einer kurzen Verzögerung (`delay(50)`), die es dem DAC ermöglicht, sich zu stabilisieren, und dem ADC, eine Messung vorzunehmen, wird der ADC-Wert ausgelesen.
   - Validierung der übertragenen Bits: Nach dem Lesen der ADC-Werte für beide Teile des Manchester-Bitpaares wird eine Überprüfung durchgeführt. Wenn der zweite ADC-Wert kleiner als der erste ist, wird er als '1' interpretiert, andernfalls als '0'. Diese Interpretation wird dann mit dem Bit der ursprünglichen Sequenz verglichen, und bei einer Abweichung wird der Fehlerzähler erhöht.
3. Ausgabe der Ergebnisse: Nachdem die gesamte Sequenz verarbeitet wurde, werden der Wert des Potentiometers (`potiValue`), der für das Wasserzeichen verwendete DAC-Wert (`Vwm`) und die Anzahl der Fehler ausgegeben. Dies ermöglicht eine Bewertung der Übertragungsqualität innerhalb der zuvor definierten Bereiche für V_S und V_{wm}.

Das Ergebnis der durchgeführten Auswertung wird im Abschnitt 5.1 anhand der seriellen Schnittstellenausgabe dargestellt und ausgewertet.

```
void transmitSequence(int potiValue, int Vwm) {
  int errors = 0;
  for (int i = 0; i < encodedSequence.length(); i += 2) {
          char originalBit = originalSequence[i / 2];
          int dacValue1 = baselineDacValue +
           ↪  (encodedSequence.charAt(i) == '0' ? -Vwm : 0);
          dacWrite(DAC, dacValue1);
          delay(50);
          int adcReading1 = analogRead(ADC);
          int dacValue2 = baselineDacValue +
           ↪  (encodedSequence.charAt(i + 1) == '0' ? -Vwm : 0);
        dacWrite(DAC, dacValue2);
        delay(50);
        int adcReading2 = analogRead(ADC);
        if (adcReading2 < adcReading1){
                bitValidation = '1';
        } else {
                bitValidation = (adcReading2 == adcReading1) ? '1'
                 ↪  : '0'; }
        if (originalBit != bitValidation)
        { errors++; }
  }
  Serial.print("PotiValue: ");
  Serial.print(potiValue);
  Serial.print(", Vwm: ");
  Serial.print(Vwm);
  Serial.print("Bit, Errors: ");
  Serial.println(errors);
}
```

Listing 12: Codeausschnitt: Wasserzeichenlogik der Evaluierung

3.5 Implementierung WebSocket-Server

In der modernen Webentwicklung spielen Echtzeit-Kommunikationsmechanismen eine zentrale Rolle, um interaktive und reaktionsfähige Anwendungen zu ermöglichen. Ein solcher Mechanismus ist der Einsatz von WebSocket-Servern, die einen bidirektionalen Vollduplex-Kommunikationskanal zwischen einem Client und einem Server aufbauen. Im Gegensatz zu herkömmlichen HTTP-Verbindungen, die unidirektional und zustandslos sind, erlauben WebSockets das Senden von Nachrichten zu jedem Zeitpunkt von beiden Enden der Verbindung. [31, S. 1] Diese Eigenschaften machen die Wahl eines WebSocket-Servers für das HAS-Gateway zu einer vielversprechenden Wahl.

Wie im Abschnitt 3.1.4 beschrieben, wird der hier behandelte WebSocket-Server auf einem Raspberry Pi 4 gehostet und in Python entwickelt. Diese Wahl von Hard- und Software bietet eine leistungsfähige Plattform für das Hosting eines Echtzeit-Kommunikationssystems. Die in Python geschriebene Serveranwendung nutzt verschiedene Bibliotheken, deren Funktionalitäten im Kapitel Abschnitt 3.2.2 besprochen wurden, um den WebSocket-Server effizient implementieren zu können.

Aufgrund der Komplexität des WebSocket Server Codes wird dieser im weiteren Verlauf des Kapitels in kleinere Segmente aufgeteilt und nur noch grob beschrieben.

3.5.1 Datenbank: Setup und Klassenstruktur

Bei der Implementierung des WebSocket-Servers spielt die Datenverwaltung eine wesentliche Rolle. Dieser Abschnitt beschreibt die Initialisierung der Datenbank und die Definition der Datenmodelle, die zur Speicherung der Daten verwendet werden. Die Datenbank und die Modelle sind zentral für die Erfassung und Analyse der Sensor- und Sequenzdaten, die vom Server verarbeitet werden.

Initialisierung der Datenbankverbindung

Die Initialisierung der Datenbankverbindung beginnt im Codeausschnitt 13 mit der Definition der Datenbank-URI, über die der Server auf die Datenbank zugreifen kann:

```
DATABASE_URI = 'sqlite:///thermistor_readings.db'
```

Wie im Code 13 zu sehen ist, wird SQLite als Datenbanksystem verwendet, und die Datenbankdatei wird als *thermistor_readings.db* benannt. Anschließend wird die Base-Klasse von SQLAlchemys Deklarativem System verwendet, um eine Basis für die Modelldeklarationen zu schaffen. Kurz gesagt wird die Base Klasse

verwendet, um sie an die selbst erstellten Python Datenbankklassen zu vererben, die automatisch Datenbankfelder den Eigenschaften der Klassen zuweisen. Die Engine, die für die Verbindung zur Datenbank verantwortlich ist, wird durch den Aufruf von *create_engine* mit der DATABASE_URI erstellt. Eine scoped Session wird eingerichtet, um eine thread-sichere Session für die gesamte Anwendung zu gewährleisten. [27]

Deklarierung der Datenmodelle

Innerhalb des Codes werden zwei Hauptdatenmodelle definiert: *Sequence* und *ThermistorReading*.

Das Sequence-Modell

Das *Sequence*-Modell wird verwendet, um Bitfolgen zu speichern, die für die Authentifizierung und die Darstellung des Wasserzeichens benötigt werden. Es enthält die Felder *id*, *sequence*, *received_bits* und *verified*. *id* dient als Primärschlüssel, während *sequence* eine vom Server zufällig generierte Bitfolge enthält. *Received_bits* enthält die vom Client gemessenen Wasserzeichenbits und *verified* gibt an, ob die BER der empfangenen Bitfolge innerhalb des Thresholds liegt.

Das ThermistorReading-Modell

Das *ThermistorReading*-Modell wird zur Speicherung von Sensormesswerten verwendet. Es besteht aus den Feldern *id*, *value* und *sequence_id*. Dabei ist *sequence_id* ein Fremdschlüssel, der eine Beziehung zum *Sequence*-Modell herstellt, um anzugeben, zu welcher Sequenz die Temperaturmessung gehört.
Die Beziehung zwischen *Sequence* und *ThermistorReading* wird durch die Funktion *relationship* definiert, die bidirektionale Beziehungen zwischen den Modellen ermöglicht. Dies erleichtert den Zugriff auf die zugehörigen Messwerte aus jeder Sequenz und umgekehrt

Erstellung der Datenbanktabellen

Die Funktion *create_tables* ist für die Erstellung der Datenbanktabellen auf der Grundlage der definierten Modelle verantwortlich. Dies geschieht durch den Aufruf von *Base.metadata.create_all*, wodurch die Tabellen im Kontext der angegebenen Engine erstellt werden. Diese Funktion wird vor der ersten Anfrage an den Server aufgerufen, um sicherzustellen, dass die Datenbankstruktur korrekt initialisiert ist.

```
DATABASE_URI = 'sqlite:///thermistor_readings.db'
Base = declarative_base()
engine = create_engine(DATABASE_URI)
db_session = scoped_session(sessionmaker(autocommit=False,
    ↪ autoflush=False, bind=engine))

class Sequence(Base):
        __tablename__ = 'sequence'
        id = Column(Integer, primary_key=True)
        sequence = Column(String(16))
        received_bits = Column(String(16), default="")
        verified = Column(Boolean, default=False)

class ThermistorReading(Base):
        __tablename__ = 'thermistor_reading'
        id = Column(Integer, primary_key=True)
        value = Column(Integer)
        sequence_id = Column(Integer, ForeignKey('sequence.id'))
        sequence = relationship("Sequence",
         ↪ back_populates="readings")

Sequence.readings = relationship("ThermistorReading",
    ↪ order_by=ThermistorReading.id, back_populates="sequence")

# Equivalent to Flask's before_first_request
async def create_tables():
        Base.metadata.create_all(bind=engine)
```

Listing 13: Codeausschnitt: Datenbank: Setup und Klassenstruktur

3.5.2 Websocket Handler

In den folgenden Abschnitten wird die Implementierung von zwei zentralen Funktionen des WebSocket Servers beschrieben. Diese Funktionen sind für die Kommunikation mit den Clients und die Verwaltung der Daten bezüglich der generierten Bitfolgen und der Temperaturmessungen verantwortlich

WebSocket Handler 1: Generierung und Übertragung einer Sequenz

Der erste Handler, dargestellt im Codeausschnitt 14, umfasst die Generierung einer zufälligen Bitsequenz und deren Übermittlung an den Client.

1. **Generierung einer zufälligen Bitfolge:** Die Funktion `generate_random_sequence()` erzeugt eine zufällige Bitsequenz der Länge 256, die aus den Zeichen '0' und '1' besteht. Diese Sequenz wird als Teil der Antwort des Servers verwendet. Die Länge der Bitfolge kann hier beliebig verändert werden.
2. **Verarbeitung eingehender Nachrichten:** Die asynchrone Funktion `handle_socket()` wartet auf Nachrichten vom WebSocket-Client. Beim Empfang einer Nachricht wird diese als JSON-Objekt interpretiert.
3. **Behandlung von Anfragen die eine neue Sequenz fordern:** Enthält die Nachricht eine Anfrage nach einer neuen Sequenz (`'new_sequence_request'`), generiert der Server eine zufällige Bitsequenz, speichert diese in der Datenbank und sendet eine Antwort mit der generierten Sequenz und ihrer ID an den Client zurück.

WebSocket Handler 2: Verarbeitung von Bit- und Temperaturdaten

Der zweite Handler, dargestellt im Codeausschnitt 15, behandelt die Verarbeitung und Validierung von empfangenen Bitdaten sowie die Speicherung von Temperaturmesswerten.

1. **Validierung der Sequenz:** Beim Empfang einer Nachricht, die ein Bit, eine Sequenz-ID und einen Temperaturwert enthält, wird die entsprechende Sequenz aus der Datenbank abgerufen. Die empfangenen Bits werden in die Spalte *received_bits*, in Höhe der angegebenen Sequenz-ID, geschrieben. Sobald alle Wasserzeichenbits empfangen wurden, validiert der Server die Sequenz, indem er die Anzahl der Nichtübereinstimmungen zwischen der gesendeten und der empfangenen Sequenz berechnet. Abhängig von der Anzahl der erlaubten Nichtübereinstimmungen (`allowed_mismatches`) wird

die Sequenz als verifiziert oder nicht verifiziert markiert. Die Anzahl der erlaubten Nichtübereinstimmungen für das entwickelte Konzept beträgt 7. Da für den verwendeten DAC-Wert für das Wasserzeichen im Ergebnis der Parameterauswertung, zu sehen in Abschnitt 5.1, eine maximale BER von 2,5% für eine Bitfolge der Länge 256 ermittelt wurde. 2,5% von 256 ergibt gerundet einen Wert von 7.

2. **Speicherung der Temperaturmessungen:** Unabhängig vom Validierungsprozess werden die empfangenen Sensorwerte zusammen mit der Sequenz-ID als neue `ThermistorReading`-Einträge in der Datenbank gespeichert.
3. **Benachrichtigung des Clients:** Nach der Validierung sendet der Server eine Statusaktualisierung an den Client, die angibt, ob die Sequenz verifiziert wurde oder nicht. Die Statusaktualisierung wird ausgesendet, um den Clients eine Reaktion im Angriffsfall zu ermögliche. Im Zuge des entwickelten Konzeptes wurde keine Reaktion definiert, da eine geeignete Reaktion abhängig vom Typ des Clients ist und somit nicht relevant für unser allgemein gehaltenes Konzept ist.

Zusammenfassend bieten diese Handler eine effiziente Methode zur Verwaltung der Echtzeitkommunikation zwischen dem Server und den Clients. Sie ermöglichen die Generierung und Validierung von Bitfolgen für das Wasserzeichen sowie die Erfassung und Speicherung von Sensormesswerten.

```
def generate_random_sequence():
        return ''.join(np.random.choice(['0', '1'], size=256))

async def handle_socket(websocket, path):
        await create_tables()  # Ensure tables are created before handling requests
        async for message_str in websocket:
                if message_str is not None:
                message = json.loads(message_str)
                if 'request' in message and message['request'] == 'new_sequence_request':
                        sequence = generate_random_sequence()
                        new_sequence_obj = Sequence(sequence=sequence)
                        db_session.add(new_sequence_obj)
                        db_session.commit()
                        response = {'type': 'sequence_update', 'sequence': sequence, 'sequence_id': new_sequence_obj.id}
                        await websocket.send(json.dumps(response))
```

Listing 14: Codeausschnitt: Websocket Handler 1

```
elif 'bit' in message and 'sequence_id' in message and 'temperature' in message:
        sequence_id = message['sequence_id']
        bit = message['bit']
        temperature = message['temperature']
        sequence = db_session.get(Sequence, sequence_id)
        if sequence:
                sequence.received_bits += bit
                if len(sequence.received_bits) == len(sequence.sequence):
                        allowed_mismatches = 7
                        mismatches = sum(1 for orig_bit, received_bit in zip(sequence.sequence, sequence.received_bits) if orig_bit != received_bit)
                        sequence.verified = mismatches <= allowed_mismatches
                        db_session.commit()
                        status_update = {'type': 'status_update', 'verified': sequence.verified}
                        await websocket.send(json.dumps(status_update))
                new_reading = ThermistorReading(value=temperature, sequence_id=sequence_id)
                db_session.add(new_reading)
                db_session.commit()
else:
        break
```

Listing 15: Codeausschnitt: Websocket Handler 2

3.5.3 Initialisierung der Webseite und Server Start

Die folgenden Abschnitte behandeln die Initialisierung der Server-Webseite und den Server-Startvorgang, einschließlich der Konfiguration der HTTP- und WebSocket-Kommunikation.

Initialisierung der Webseite

Im Codeausschnitt 16 wird die Initialisierung der Webseite und die Dynamisierung der Inhalte basierend auf den aktuellen Sensordaten implementiert.

1. **Abrufen der neuesten Sequenz und Temperaturwerte:** Die asynchrone Funktion `index` dient als Handler für HTTP GET-Anfragen auf die Root-URL ('/'). Sie holt die neueste Sequenz aus der Datenbank und extrahiert den zuletzt aufgezeichneten Temperaturwert sowie den Verifikationsstatus der Sequenz.
2. **Dynamische Anpassung des HTML-Inhalts:** Der HTML-Inhalt der Webseite wird aus einer Template-Datei (*index.html*) geladen. Platzhalter im HTML-Code für Temperatur und Status werden durch die tatsächlichen Werte aus der Datenbank ersetzt.
3. **Rückgabe der angepassten Webseite:** Der angepasste HTML-Inhalt wird als Antwort auf die HTTP-Anfrage zurückgegeben, wobei der MIME-Typ 'text/html' angegeben wird, um den Browser anzuweisen, den Inhalt entsprechend zu interpretieren und anzuzeigen.

Server Start

Der Codeausschnitt 17 beschreibt, wie der Server für HTTP- und WebSocket-Anfragen gestartet wird.

1. **Konfiguration des HTTP-Servers:** Eine Instanz der Klasse `web.Application` wird erzeugt, um die Webanwendung zu repräsentieren. Der `index`-Handler wird für die Root-URL konfiguriert, gefolgt von der Einrichtung und dem Start des HTTP-Servers auf Port 8080, wodurch die Webseite für Benutzer zugänglich gemacht wird.
2. **Start des WebSocket-Servers:** Parallel zum HTTP-Server wird ein WebSocket-Server auf Port 5000 gestartet. Dieser verwendet die Funktion `handle_socket`, um WebSocket-Verbindungen und -Nachrichten zu verarbeiten.
3. **Permanente Ausführung:** Nach dem Start beider Server geht das Programm in einen Wartezustand ein, der durch `await asyncio.Future()` definiert wird, um dauerhaft auf Anfragen warten zu

können.

Zusammenfassend bieten diese Implementierungen eine komplette Serverlösung, die sowohl statische Webinhalte dynamisch generieren und ausliefern als auch Echtzeitkommunikation über WebSockets unterstützen kann.

```
async def index(request):
        sequence =
        ↪  db_session.query(Sequence).order_by(Sequence.id.desc()).first()
        if sequence:
                temperature = sequence.readings[-1].value
                status = "OK" if sequence.verified else "Malicious
                ↪  Activity"
        else:
                temperature = None
                status = "No data available"
        with open("templates/index.html", "r") as f:
                html_content = f.read()
        html_content = html_content.replace("{{ temperature }}",
        ↪  str(temperature))
        html_content = html_content.replace("{{ status }}", status)
        return aiohttp.web.Response(text=html_content,
        ↪  content_type='text/html')
```

Listing 16: Codeausschnitt: Initialisierung der Webseite

```python
async def start_server():
        app = web.Application()
        app.router.add_get('/', index)
        runner = web.AppRunner(app)
        await runner.setup()
        try:
                site = web.TCPSite(runner, "0.0.0.0", 8080)
                await site.start()
                print("HTTP server started on
                 ↪  http://0.0.0.0:8080/")
        except Exception as e:
                print(f"Error starting HTTP server: {e}")
        async with websockets.serve(handle_socket, "0.0.0.0",
         ↪  5000):
                print("WebSocket server started on
                 ↪  ws://0.0.0.0:5000/")
                await asyncio.Future()  # Run forever

if __name__ == "__main__":
        asyncio.run(start_server())
```

Listing 17: Codeausschnitt: Server Start

3.6 Finales Konzeptes

Diese Sektion erläutert das entwickelte Konzept und die Implementierungsdetails der Wasserzeichenlogik, die darauf ausgelegt ist, die Sicherheit in HAS und IoT-Geräten zu verbessern. Zwei Schemata untermauern diese Erläuterung: der Server, dargestellt in Abbildung 3.8, und der Client, zu sehen in Abbildung 3.7. Diese illustrieren die wesentlichen Komponenten und ihre Funktionalitäten innerhalb des Systems.

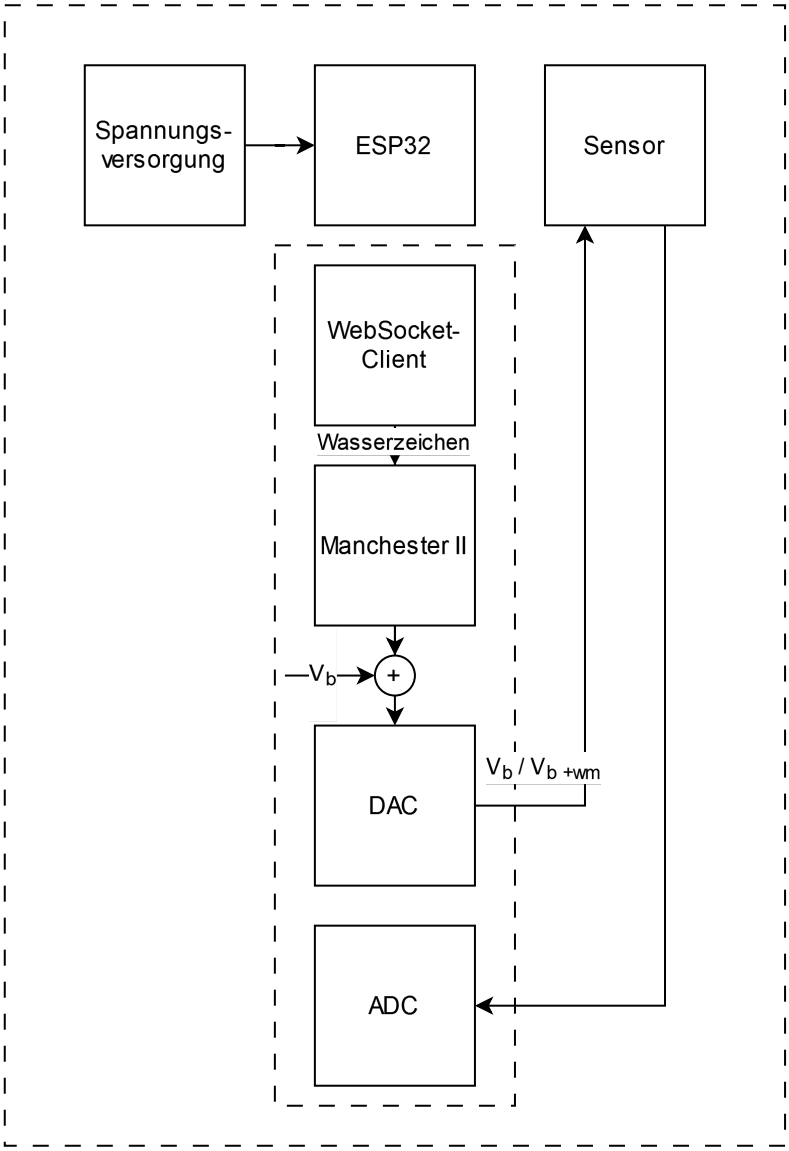

Abbildung 3.7: Client Konzept

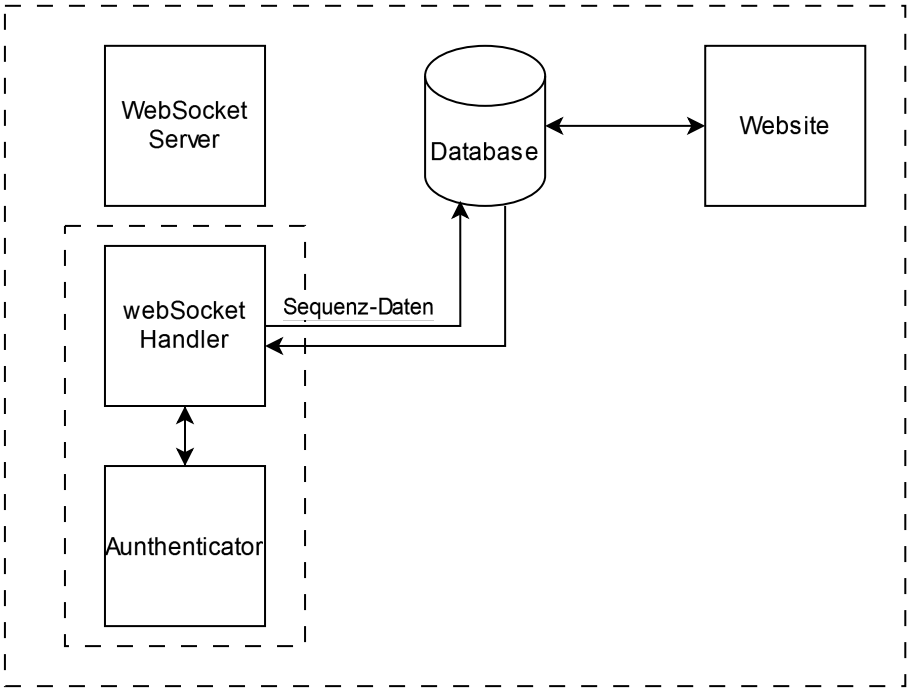

Abbildung 3.8: Server Konzept

3.6.1 konzeptioneller Workflow

Dieses Kapitel beschreibt einen Beispielhaften Workflow des entwickelten Konzepts.

1. Initiierung des WebSocket-Servers und der Webseite sowie Erstellung der Datenbanken, falls diese noch nicht existieren.
2. Der Server wartet anschließend auf eine Kommunikation von einem Client.
3. Der Client wird aktiviert und stellt eine Verbindung zum WLAN her.
4. Der Client versucht, eine Verbindung zum WebSocket-Server herzustellen.
5. Nach erfolgreicher Verbindung fordert der Client eine Wasserzeichen-Bitsequenz an.
6. Der Server generiert eine neue Sequenz, speichert diese in der Sequenzdatenbank mit einer zugeordneten ID und sendet sie inklusiver der ID an den Client.
7. Unter Verwendung der Manchester-II-Kodierung kodiert der Client die erhaltene Wasserzeichen-Sequenz.
8. Der Client wendet über den DAC eine Spannung V_{b} auf den Sensor an, wenn das aktuelle kodierte Bit 1 ist, oder $V_{\mathrm{b+wm}}$, wenn es 0 ist.
9. V_{S} wird dann vom Client am ADC gelesen und in einer Variablen gespeichert.
10. Der Client wiederholt die Schritte 9 und 10, um beide Bits des Manchester-Paares zu erhalten.

11. Der Client bewertet, ob der erste oder der zweite Wert höher ist; ist der letztere niedriger, entspricht dies dem Manchester-Paar 10, und somit werden die beiden Messwerte als Wasserzeichen-Bit mit dem Wert 1 klassifiziert.
12. Der Client sendet die Sequenz-ID, das klassifizierte Wasserzeichen-Bit und den Sensorwert ohne Bias zurück an den Server.
13. Nach Erhalt der Daten speichert der Server diese in den entsprechenden Datenbanken.
14. Wenn dem Client nur noch 40 Bits der kodierten Sequenz verbleiben, fordert er eine neue an um Wartezeiten nach Abschluss einer Sequenz zu vermeiden.
15. Eine neue Sequenz wird vom Server auf Anfrage an den Client gesendet.
16. Der Server führt anschließend den Verifizierungsprozess durch und vergleicht die empfangenen Bits mit den Sequenzbits, die er ursprünglich an den Client gesendet hat.
17. Die Verifizierung gilt als gescheitert, wenn die BER die Schwelle von 2 Prozent überschreitet.

3.6.2 Client WebSocket Kommunikation

In diesem Kapitel wird die WebSocket Kommunikation aus Client Seite anhand der Implementierten Codeausschnitte eines Switch-Cases erklärt.

```
1 case WStype_DISCONNECTED:
2 Serial.println("[WebSocket] Disconnected");
3 connected = 0;
4 break;
```

Listing 18: Codeausschnitt: Client Disconnected

Client Disconnected

- Der Client erreicht den Zustand `WStype_DISCONNECTED`, wenn die WebSocket-Verbindung getrennt wird oder ein Verbindungsversuch fehlschlägt. Dies kann durch Netzwerkprobleme oder einen Serverausfall verursacht werden.
- Eine Meldung wird auf der seriellen Konsole ausgegeben, um den Verbindungsverlust anzuzeigen: `[WebSocket] Disconnected`.
- Die Variable connected wird auf 0 gesetzt, was anzeigt, dass keine aktive Verbindung besteht.

```
void requestNewSequence() {
        webSocket.sendTXT("{\"request\":
        ↪  \"new_sequence_request\"}");
        Serial.println("Requested new sequence.");
}

case WStype_CONNECTED:
Serial.println("[WebSocket] Connected");
connected = 1;
requestNewSequence();
break;
```

Listing 19: Codeausschnitt: Client Connected

Client Connected

- Tritt ein, wenn eine WebSocket-Verbindung erfolgreich hergestellt wurde `WStype_CONNECTED`.
- Eine Bestätigungsnachricht wird auf der seriellen Konsole ausgegeben: `[WebSocket] Connecte`
- Die Variable `connected` wird auf 1 gesetzt, was eine bestehende Verbindung kennzeichnet.
- Unmittelbar nach der Verbindungsaufnahme fordert der Client eine neue Wasserzeichen-Bitsequenz an, indem die Funktion `requestNewSequence()` aufgerufen wird.
- Die Funktion `requestNewSequence()` verwendet den Befehl `webSocket.sendTXT(¨¨);`, um eine Nachricht an den Server zu senden.

Client Text Received

- Wird ausgelöst, wenn der Client eine Textnachricht über die WebSocket-Verbindung empfängt `WStype_TEXT`.
- Zunächst wird die empfangene Nachricht auf der seriellen Konsole protokolliert: `[WebSocket] Text received`.
- Die empfangene Nachricht, die im JSON-Format erwartet wird, wird deserialisiert, um auf die enthaltenen Daten zugreifen zu können.
- Überprüft, ob die Nachricht eine Wasserzeichen-Sequenz `sequence` enthält. Falls ja, wird die Sequenz extrahiert und für die Manchester-II-Kodierung vorbereitet.
- Die Manchester-II-Kodierte Sequenz wird dann basierend auf dem aktuellen Zustand des Clients als

```
case WStype_TEXT:
Serial.println("[WebSocket] Text received");
DynamicJsonDocument doc(1024);
deserializeJson(doc, payload);
if(doc.containsKey("sequence")) {
        String newSequence = doc["sequence"].as<String>();
        String encodedSequence = manchesterEncode(newSequence);
        Serial.print("Received Sequence: ");
        Serial.println(newSequence);
        if (!currentSequence.isProcessing) {
                currentSequence.sequence = encodedSequence;
                currentSequence.sequenceId =
                 ↪  doc["sequence_id"].as<int>();
                currentSequence.isProcessing = true;
                bitIndex = 0;
        } else {
                nextSequence.sequence = encodedSequence;
                nextSequence.sequenceId =
                 ↪  doc["sequence_id"].as<int>();
                nextSequence.isProcessing = false;
        }
}
break;
```

Listing 20: Codeausschnitt: Client Text received

aktuelle oder nächste Sequenz festgelegt.

- Für den Fall, dass es sich um die erste empfangene Sequenz (`!currentSequence.isProcessing`) handelt und somit noch keine Sequenz verarbeitet wird, wird sie als die aktuelle Sequenz klassifiziert und der bitIndex wird auf 0 zurückgesetzt.
- Erhält der Client während der Verarbeitung einer Sequenz eine weitere Sequenz, wird diese als nächste Sequenz vorgemerkt und markiert (`isProcessing = false`), um nach Abschluss der aktuellen Sequenz verarbeitet zu werden.

Für die Sendung der Daten wird in der Wasserzeichen Logik nach Identifikation eine Nachricht über den Befehl zu sehen im Codeausschnitt 21, mit der aktuell bearbeitenden Sequenz ID, dem identifizierten Wasserzeichenbit und der aktuellen Sensormessung an den Server gesendet.

```
String message = "{\"sequence_id\": " +
    ↪ String(currentSequence.sequenceId) + ", \"bit\": \"" +
    ↪ String(bitToSend) + "\", \"temperature\": " +
    ↪ String(temperatureReadingActual) + "}";
```

Listing 21: Codeausschnitt: Sensordatenübermittlung

4 Angriffsszenarien

In diesem Kapitel werden drei Angriffsszenarien untersucht, um die Funktionsfähigkeit des sensorseitigen Wasserzeichenüberprüfungskonzepts außerhalb des Normalbetriebs zu testen. Die Angriffe werden in den folgenden Unterkapiteln detailliert beschrieben. Jedes Unterkapitel enthält zusätzlich ein Schema der Testszenarien sowie die notwendigen Code-Snippets, um die Reproduzierbarkeit zu gewährleisten. Die Ergebnisse der Tests werden im Abschnitt Kapitel 5 Resultate diskutiert, analysiert und dargestellt.

4.1 Grounding Attack

4.1.1 Definition des Angriffsszenarios

Ein Grounding-Angriff, auch Short Circuit-Angriff genannt, ist ein Angriff, der in die Kategorie der sogenannten Fault-Injection-Angriffe fällt. Beispiele für diverse Arten der Fault-Injection Angriffe finden sich in einer Vielzahl von Arbeiten wieder.[32]–[36] Konkret kann dieser Angriff verwendet werden, um fehlerhaftes Verhalten im Zielsystem auszulösen. Ebenso wird dieser Angriff teilweise verwendet, um Systeminstruktionen wie z.B. Sicherheitsüberprüfungen zu überspringen.[37]–[40] Unter anderem veröffentlichte der Autor LimitedResults in seinem Blog [41] einen Artikel, in dem er einen Grounding-Angriff verwendet, um erfolgreich einen APPROTECT-Bypass auf einem geschützten nRF52840-System durchzuführen. Dieser Bypass hat zu Folge, dass der Zugriff auf eine voll funktionsfähige Debugging-Schnittstelle ermöglicht wird.

4.1.2 Umsetzung des Szenarios

Für die praktische Umsetzung des Grounding Attack Szenarios wurde der in Abbildung 3.5 dargestellte Testaufbau verwendet. Für das Angriffsszenario bildet der Spannungsteiler den Kern des Aufbaus, in dem das digitale Potentiometer die Schlüsselrolle spielt. Durch Einstellen des Potentiometers auf 0 Ohm simulieren wir den Grounding- oder Short Circuit-Angriff, indem wir das Signal, das normalerweise am Ausgang des Spannungsteilers und damit am Eingang des ADC anliegen würde, effektiv auf Masse ziehen. Dieser Zu-

stand führt dazu, dass das Zielsystem keine echten Spannungsmessungen mehr durchführen kann, wodurch das Sensorsignal aktiv manipuliert wird.

4.1.3 Implementierung der Simulation

Der folgende Codeausschnitt 22 zeigt die Implementierung des Grounding-Angriffs im Rahmen der Simulation. Der Angriff wird durch eine einfache logische Bedingung innerhalb der `loop()`-Funktion des Mikrocontrollers gesteuert.

1. **Definition des Angriffsauslösers:** Der als `BUTTON` definierte Pin 0 wird dem internen Boot Button zugeordnet. Nach dem Bootvorgang des Systems steht dieser Pin für verschiedene Funktionen zur Verfügung und dient in unserem Szenario als physischer Trigger für den Grounding-Angriff.
2. **Angriffsbedingung:** Innerhalb der `loop()`-Funktion prüft der Mikrocontroller kontinuierlich den Status des `BUTTON`-Eingangs. Wenn dieser Eingang auf LOW gesetzt ist, was einer physischen Betätigung des Buttons entspricht, wird das digitale Potentiometer auf 0 Ohm eingestellt.
3. **Manipulation des Signals:** Die Einstellung des Potentiometers auf 0 Ohm zieht das Signal, das unter normalen Umständen zum ADC gelangen würde, auf Ground. Diese Aktion simuliert den Grounding-Angriff, der darauf abzielt, das Verhalten des Zielsystems durch die Manipulation der erwarteten Eingangssignale zu stören.

```
1 #define BUTTON 0
2 void loop() {
3         ---
4         if (digitalRead(BUTTON) == LOW) {
5                 setDigitalPotentiometer(0);
6         }
7         ---
8 }
```

Listing 22: Codeausschnitt: Grounding Attack

Das Angriffsszenario demonstriert die Einfachheit, mit der ein Grounding-Angriff in einem realen Szenario durchgeführt werden kann, vorausgesetzt, der Angreifer hat physischen Zugriff auf das Zielgerät oder kann

dieses Verhalten auf andere Weise fernsteuern.

4.2 Voltage Injection

4.2.1 Definition des Angriffsszenarios

Ein Voltage Injection-Angriff ist ein weiteres Beispiel für einen Fault-Injection-Angriff, unterscheidet sich jedoch von den zuvor besprochenen Grounding-Angriffen. Bei dieser Art von Angriff wird das System manipuliert, indem eine externe Spannungsquelle in eine der Komponenten, in diesem Fall den ADC, eingespeist wird [1, S.4]. Im Gegensatz zu Software-basierten Angriffen erfordert der Voltage Injection-Angriff keine Code-Änderungen. Stattdessen wird einfach eine externe Spannungsquelle verwendet, um direkt auf die Hardware einzuwirken.

4.2.2 Umsetzung des Szenarios

Für die Umsetzung des Voltage Injection Angriffs wurde die Verbindung zum ADC des Clients modifiziert. Anstelle der normalen Verbindung zum Ausgang des Spannungsteilers wurde der ADC-Eingang direkt mit einer externen Spannungsquelle mti einer Spannung von 2 Volt verbunden.

4.3 Replay/Modification

4.3.1 Definition des Angriffsszenarios

Ein Replay/Modification-Angriff ist eine spezialisierte Form des Man-in-the-Middle-Angriffs (MitM), der in sicherheitskritischen Kommunikationssystemen, insbesondere in solchen, die Sensordaten übermitteln, ernsthafte Bedrohungen darstellen kann. In diesem Szenario greift der Angreifer physisch in die Kommunikation zwischen einem Mikrocontroller und einem Sensor ein, indem er einen eigenen, feindlichen Mikrocontroller in die Kommunikationskette einschleust.

Wie in der stark vereinfachten Abbildung 4.1 dargestellt, ist der Mikrocontroller des Angreifers so konfiguriert, dass er direkt die Werte des eigentlichen Mikrocontrollers über dessen ADC einlesen und diese Werte im Replay-Modus über einen DAC zum Sensor weiterleiten kann. In einer zweiten Phase ist der Angreifer in der Lage, über eine WiFi-Verbindung den Sensorwert zu analysieren, um im Anschluss gefälschte Sensordaten zu generieren und diese an den eigentlichen Mikrocontroller zu senden.

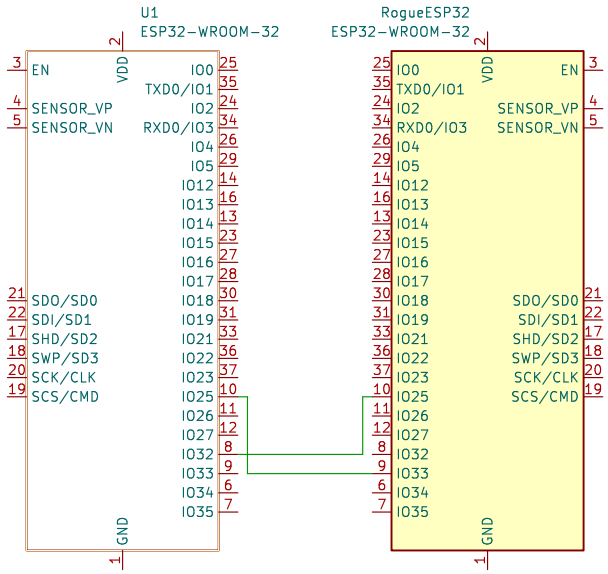

Abbildung 4.1: Vereinfachter Replay/Modification Testaufbau

4.3.2 Umsetzung des Szenarios

Der kritische Punkt dieses Angriffs ist die Fähigkeit des Angreifers, gefälschte Sensordaten so zu manipulieren, dass sie vom HAS-Gateway als authentisch wahrgenommen werden, indem sie ein korrektes Wasserzeichen enthalten. Diese Wasserzeichen-Bits werden kontinuierlich vom HAS Client an das Gateway gesendet, welches aufgrund der validen Wasserzeichen die Daten als echt verifiziert.

4.3.3 Demonstration der Angriffsmöglichkeit

Um die Machbarkeit dieses Angriffszenarios zu testen, ist keine vorherige Aufzeichnung oder Analyse der Daten notwendig. Die bloße Möglichkeit, gefälschte Daten erfolgreich und unerkannt zu übertragen, stellt bereits ein signifikantes Sicherheitsrisiko dar. Der simulierte Angriffsprozess umfasst folgende Schritte:

1. Der feindliche Mikrocontroller liest die ursprünglichen Sensordaten über seinen ADC ein.
2. Die Daten werden analysiert und manipuliert, um gefälschte, aber authentisch wirkende Sensordaten zu erzeugen.
3. Der manipulierte Wert wird dann über einen DAC des Angreifer-Mikrocontrollers an den eigentlichen Mikrocontroller gesendet.

5 Resultate

In diesem Kapitel werden die Resultate der durchgeführten Tests analysiert und diskutiert. Ziel des Kapitel ist es einen näheren Einblick in die Robustheit und Effektivität der kontinuierlichen sensorseitigen Wasserzeichenauthentifizierung zu geben.

5.1 Evaluierung der Wasserzeichen Parameter

Die Untersuchung der Wasserzeichenparameter, insbesondere der Einfluss von V_{wm} (Wasserzeichenspannung) und V_{S} (Sensorspannung), auf die Robustheit der Wasserzeichenauthentifizierung ist entscheidend für das Verständnis und die Optimierung des Systems. Die durchgeführten Tests 3.4.2, basierend auf einer 256 Zeichen langen Bitfolge, deckten Kombinationen im Bereich von $0 \leq V_{\text{wm}} \leq 0.077\text{V}$ für V_{wm} und $0 \leq V_{\text{S}} \leq 1.5\text{V}$ für V_{S} ab.

Die in Abbildung 5.1 dargestellten Ergebnisse zeigen, dass bei niedrigen Werten die BER sowohl für V_{wm} als auch für V_{S} nahe bei 50% liegt. Zu beachten ist, dass aufgrund des Sicherheitsmechanismus in der Wasserzeichenlogik für zwei ADC-Werte, die zum selben Manchester-Paar gehören und den gleichen Wert haben, ein Bitwert von 1 annehmen was einen Bias in Richtung von Wasserzeichen mit mehrere Bits mit Wert 1 generiert.

Das erzielte Ergebnis unterstreicht die Notwendigkeit, die Parameter V_{wm} und V_{S} sorgfältig zu wählen, um die Genauigkeit der Wasserzeichenauthentifizierung zu maximieren. Die Fähigkeit, unter variierenden Bedingungen zuverlässig zu funktionieren, ohne eine unverhältnismäßig hohe BER zu verursachen, ist für die Implementierung einer effektiven Wasserzeichenauthentifizierung in praktischen Anwendungsfällen von entscheidender Bedeutung. Für die Umsetzung des entwickelten Konzepts unter Verwendung der gleichen Hardware ergab die Analyse einen Wert von ca. 50mV für V_{wm}, bei einer Spannung V_{S} von weniger als 1,2V mit einer Toleranz von 2%, bei einer Bitlänge von 256.

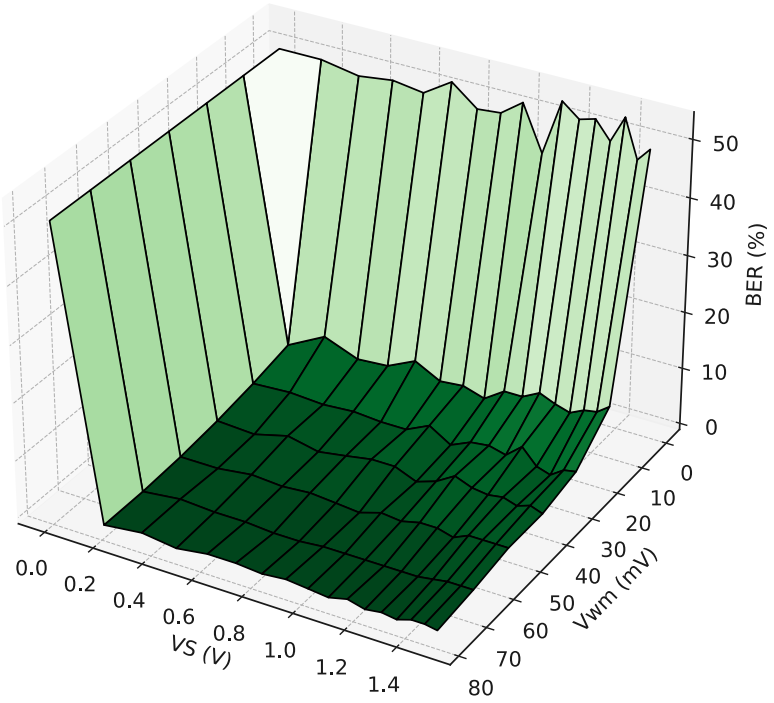

Abbildung 5.1: Wasserzeichen Evaluierung

5.2 Normalbetrieb

Die Evaluierung des Systems unter Standardbedingungen konzentriert sich insbesondere auf die Effizienz und Zuverlässigkeit der Wasserzeichen-Authentifizierung unter normalen Betriebsbedingungen. Bei einer festgelegten V_{wm} von 50mV zeigen die in Abbildung 5.2 visualisierten Ergebnisse, wie die Authentifizierungsmethode unter alltäglichen Betriebsbedingungen funktioniert.

Interessanterweise wurde nur bei der niedrigsten getesteten Potentiometereinstellung von 6250 Ohm eine leicht erhöhte BER festgestellt. Dies deutet darauf hin, dass bereits eine Spannungsmodulation von 50mV ausreicht, um das in 3.1 zu sehende Grundrauschen zu überdecken. Dies bestätigt, dass diese Modulation im Normalbetrieb keine signifikanten Störungen verursacht und die Authentifizierungsmethode robust gegenüber den üblichen Grundrauschen ist.

Die Ergebnisse sind besonders relevant, da sie zeigen, dass die Wasserzeichen-Authentifizierung nicht nur ein hohes Maß an Sicherheit gegen potenzielle Angriffe bietet, sondern auch den normalen Betrieb des Systems nicht beeinträchtigt. Die Fähigkeit, Sicherheitsmaßnahmen zu implementieren, die das System nicht stören oder seine Leistung beeinträchtigen, ist entscheidend für die Akzeptanz und den Einsatz in realen Anwendungen.

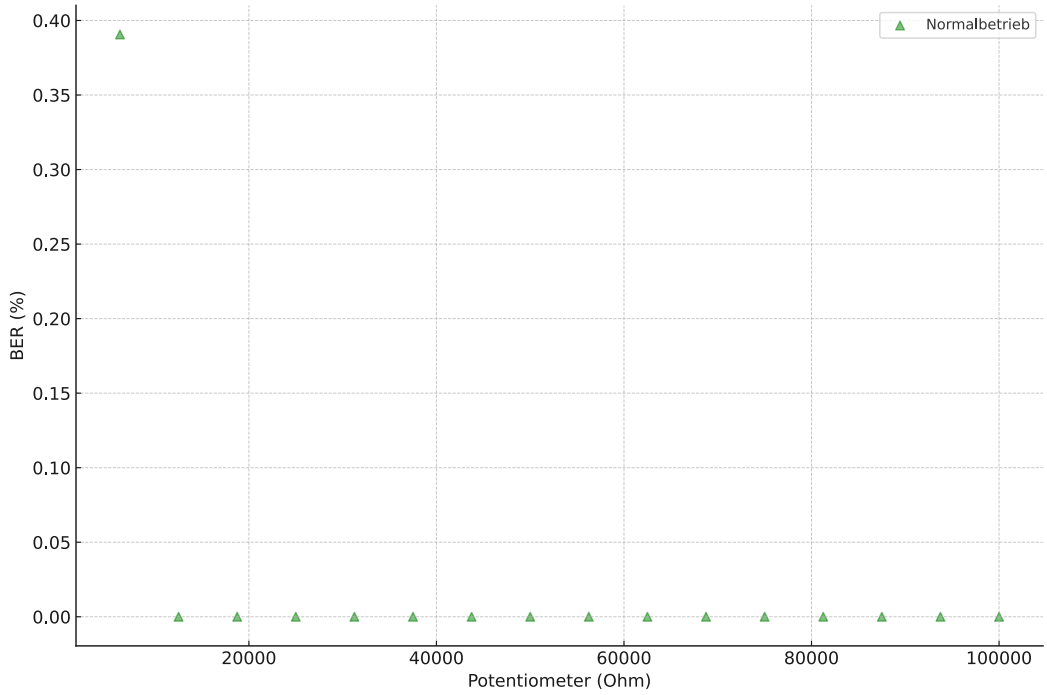

Abbildung 5.2: BER im Normalbetrieb

5.3 Grounding Attack

Die Analyse des Systems unter dem Einfluss eines Grounding-Angriffs zeigt erwartungsgemäß eine BER von ca. 50% für eine gleichverteilte Sequenz von Wasserzeichenbits. Die signifikanten Auswirkungen des in 3.1 dargestellten Grundrauschens des ADC werden in Abbildung 5.3 deutlich.

In einem hypothetischen Szenario ohne Grundrauschen würde man eine konstante BER von 50% erwarten,

basierend auf der Annahme, dass der ADC kontinuierlich den Wert 0 liest. Gemäß der implementierten Wasserzeichenlogik wird, wenn zwei Messwerte eines Manchester-II-Paares identisch sind, standardmäßig der Bitwert 0 zugewiesen. Da genau die Hälfte der Wasserzeichenbits in unserer gleichverteilten Testbitfolge den Wert 0 hat, würden dementsprechend 50% der Wasserzeichenbits korrekt interpretiert werden. Folglich sind alle Abweichungen von diesem 50%-Wert direkt auf das Grundrauschen des ADC zurückzuführen.

Dieses Ergebnis unterstreicht die Bedeutung des Grundrauschens bei der Analyse von Sicherheitssystemen, insbesondere bei der Authentifizierung von Wasserzeichen. Es zeigt, dass selbst in einem Szenario, in dem die direkten Auswirkungen eines Angriffs theoretisch vorhersagbar sind, die physikalischen Eigenschaften der Hardware (in diesem Fall das Grundrauschen des ADC) einen signifikanten Einfluss auf die Systemleistung haben können. Somit liefert diese Untersuchung wertvolle Erkenntnisse für den Entwurf robusterer Authentifizierungssysteme, die externe Störfaktoren effektiv berücksichtigen.

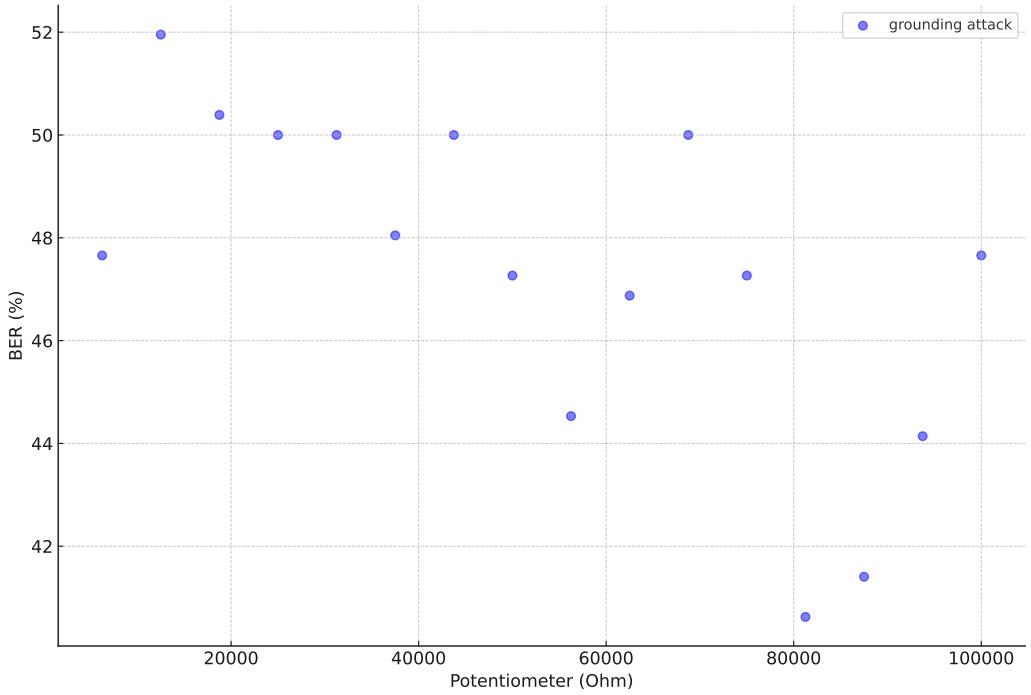

Abbildung 5.3: BER im Angriffsszenario Grounding Attack

5.4 Voltage Injection

Die Ergebnisse des Voltage Injection Angriffs, dargestellt in Abbildung 5.4, weisen darauf hin, dass dieser Angriff ähnliche Auswirkungen auf die BER hat wie der Grounding-Angriff. Einziger Unterschied hier ist die Verwendung einer externen Spannungsquelle die zusätzlich zum ADC ebenfalls ein gewisses Grundrauschen mit sich bringen kann.

Die Vergleichsanalyse in Abbildung 5.5 fasst die Ergebnisse des Normalbetriebs, des Grounding-Angriffs und der Voltage Injection zusammen und veranschaulicht die Ähnlichkeiten der Angriffsszenarien.

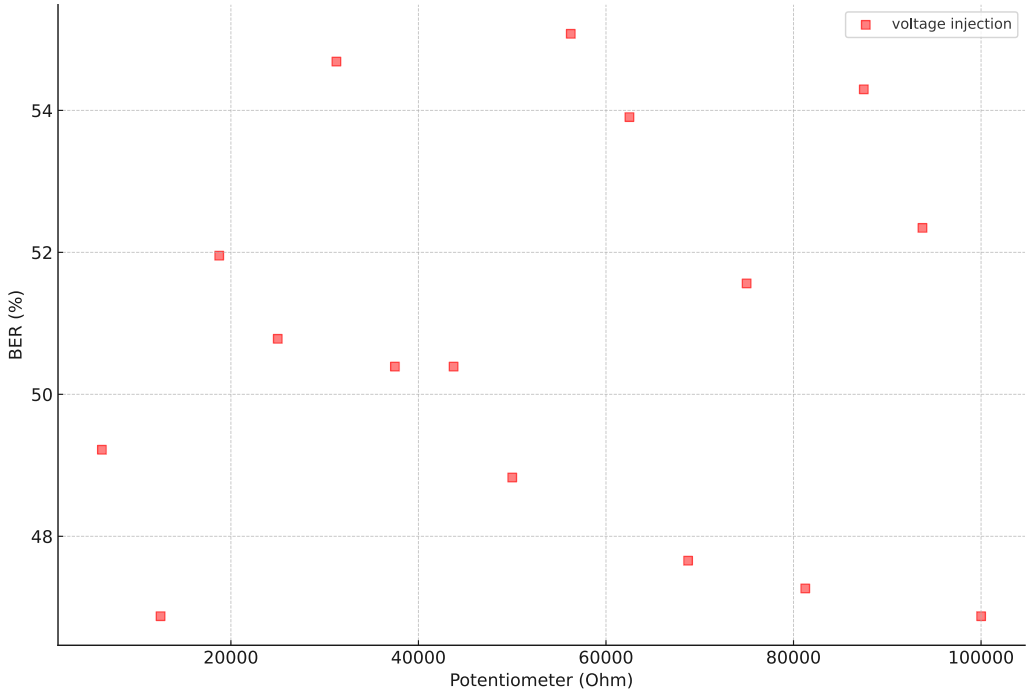

Abbildung 5.4: BER im Angriffsszenario Voltage Injection

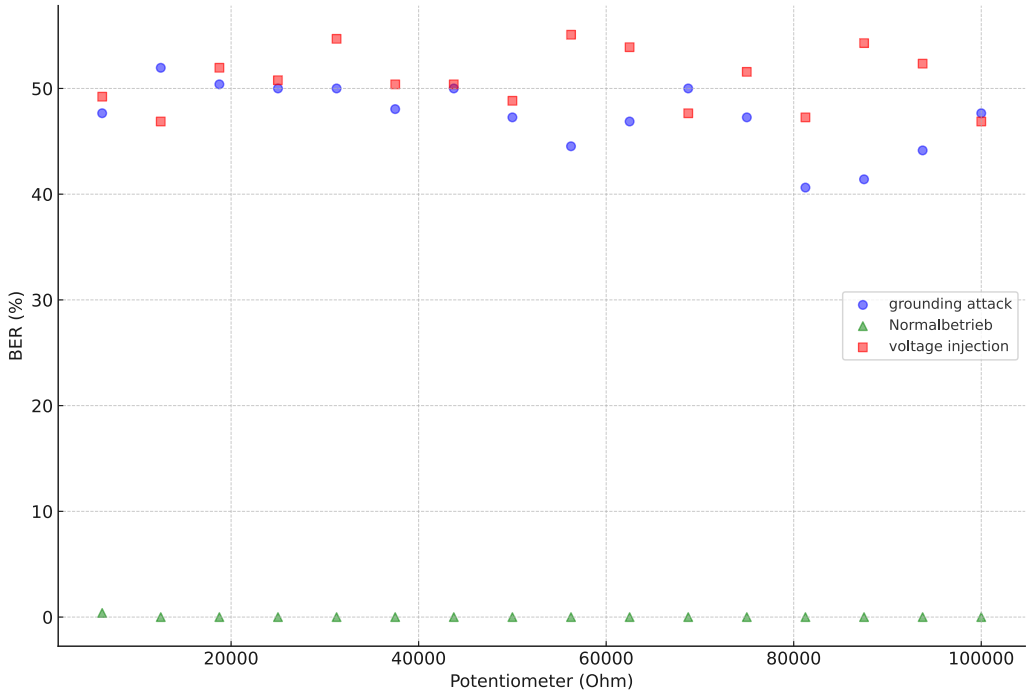

Abbildung 5.5: Wasserzeichen Evaluierung

5.5 Replay/Modification

Bei der Durchführung des Angriffsszenarios 4.3Replay/Modification wurden signifikante Herausforderungen identifiziert, die nicht in der direkten Manipulation des Wasserzeichens selbst, sondern in der Geschwindigkeit der Datenverarbeitung liegen. Es hat sich gezeigt, dass die Zeit, die benötigt wird, um das Signal des ESP32 auszulesen, das Wasserzeichen zu extrahieren und es in die manipulierten Sensordaten einzubetten, zu lang ist, um einen effektiven Angriff in Echtzeit durchzuführen.

Eine ebenfalls getestete Hypothese war, dass eine pseudosynchrone Kommunikation zwischen den Geräten durch künstlich eingefügte Wartezeiten in der Wasserzeichenlogik erreicht werden könnte. Diese Annahme wurde jedoch durch die inhärente Asynchronität der Kommunikation mit dem WebSocket Server untergraben. Die daraus resultierenden unvorhersehbaren Verzögerungen machten eine präzise Synchronisation der Geräte unmöglich, was eine wesentliche Voraussetzung für den Erfolg des Angriffs gewesen wäre.

Trotz der theoretischen Machbarkeit des Angriffs, basierend auf der Tatsache, dass nur die Differenz zwischen zwei Messungen eines Manchesterpaares zur Identifizierung des Wasserzeichenbits verwendet wird und der ausgelesene Wert selbst nicht verifiziert werden kann, stellt die zeitliche Verzögerung ein unüberwindbares Hindernis dar. Dies deutet darauf hin, dass der Angriff zwar konzeptionell durchführbar ist, aber praktische Einschränkungen seine Umsetzung in einem realen Szenario verhindern.

Aus diesen Beobachtungen lässt sich schließen, dass die Sicherheit des Systems gegen Replay/Modification-Angriffe nicht durch die Robustheit des Wasserzeichenverfahrens selbst gewährleistet wird, sondern durch die praktischen Herausforderungen bei der Umsetzung solcher Angriffe in Echtzeit.

5.6 Verhalten bei niedrigen Temperaturen

Da das entwickelte Konzept einen Thermistor als Sensor verwendet, ist es wichtig, das Verhalten des Systems auch unter extremen Bedingungen zu testen. Zu diesem Zweck wurden die Tests für den Normalbetrieb sowie für das Angriffsszenario Grounding Attack bei einer Umgebungstemperatur von ca. -17°C durchgeführt. Die Ergebnisse dieser Tests sind in Abbildung 5.6 dargestellt und zeigen einen signifikanten Anstieg der BER im Normalbetrieb.

Dieser Anstieg der Bitfehlerrate kann größtenteils auf die charakteristischen Eigenschaften des NTC-Thermistors zurückgeführt werden, dessen Widerstandswert mit sinkender Temperatur zunimmt, wie im Abschnitt 3.1.2 beschrieben. Eine Erhöhung des Thermistorwiderstandes führt zu einer deutlichen Verringerung der Ausgangsspannung V_S des Spannungsteilers, die vom ADC eingelesen wird. Durch die reduzierte Ausgangsspannung wirkt sich das Grundrauschen stärker auf die Messergebnisse aus, was zu einer Erhöhung der BER führt.

Im Gegensatz dazu zeigt das Grounding Attack Szenario aufgrund der niedrigen Temperaturen keine signifikanten Auswirkungen. Unabhängig von der Widerstandsänderung des Thermistors bleibt die Spannung aufgrund des Angriffsmechanismus gegen Masse gezogen.

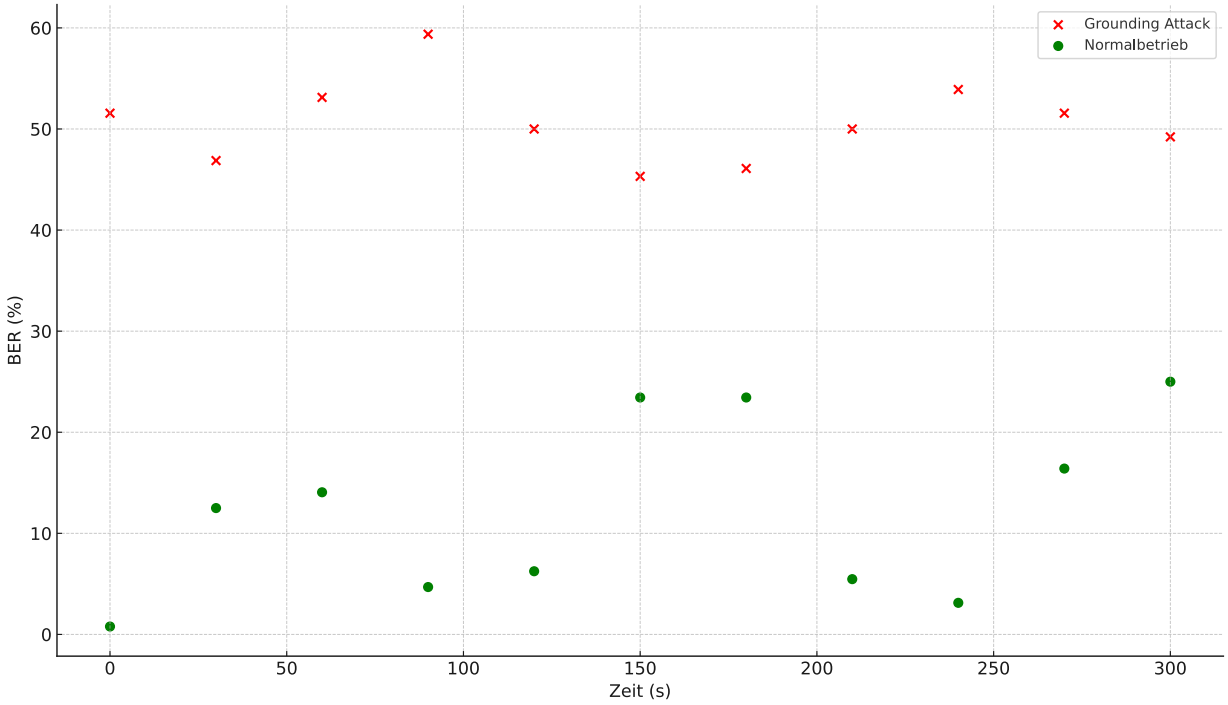

Abbildung 5.6: BER bei -17°C

6 Diskussion

Ziel dieser Diplomarbeit war die Entwicklung und Implementierung eines umfassenden Konzepts zur kontinuierlichen sensorseitigen Verifikation von Wasserzeichen. Der Schwerpunkt lag dabei auf der Untersuchung der Funktionalität unter normalen Bedingungen sowie unter spezifischen Angriffsszenarien, um die Leistungsfähigkeit und mögliche Schwachstellen des Systems zu evaluieren. Die zentralen Forschungsfragen zielten darauf ab, die Entwicklung und Implementierung eines solchen Systems zu erforschen, seine Zuverlässigkeit unter verschiedenen Bedingungen zu testen und unerwartete Schwachstellen zu identifizieren.

6.1 Diskussion der Ergebnisse

6.1.1 Konzeptentwicklung und Implementierung

Die Entwicklung des Konzepts zur kontinuierlichen Überprüfung von sensorseitigen Wasserzeichen in dieser Arbeit baut auf dem grundlegenden Wasserzeichenkonzept auf, das in der Studie *Watermarking Based Sensor Attack Detection in Home Automation Systems* [1] von Ruotsalainen et al. vorgestellt wurde. Die Umsetzung des Konzepts in der vorliegenden Arbeit erforderte eine Anpassung und Erweiterung des ursprünglichen Ansatzes, um den spezifischen Anforderungen und Herausforderungen der kontinuierlichen sensorseitigen Verifikation von Wasserzeichen gerecht zu werden. Dies beinhaltete unteranderem die Implementierung eines HAS-Gateways als zentrale Authentifizierungsstelle.

6.1.2 Herausforderungen bei der Implementierung

Die Umsetzung dieses erweiterten Konzepts war mit einer Reihe von Herausforderungen verbunden, die von technischen Hürden bis hin zu Sicherheitsbedenken reichten. Eine Schlüsselherausforderung bestand darin, eine effiziente und zuverlässige Übertragung der Wasserzeichen in Echtzeit zu gewährleisten, die an die Beschränkungen und Schwachstellen der verwendeten Hardware angepasst werden musste. Die erfolgreiche Umsetzung dieses Konzepts demonstriert das Potenzial der Verwendung von Wasserzeichen zur Erhöhung

der Sicherheit in Home Automation Systems und ähnlichen Anwendungen. Es zeigt auch, wie wichtig die kontinuierliche Weiterentwicklung und Anpassung von Sicherheitsmechanismen ist, um der sich schnell ändernden Szenerie von Cybersicherheitsbedrohungen zu begegnen.

6.1.3 Analyse der Bit Error Rate und der False Positive/Negative Raten

Ein Schlüsselaspekt dieser Arbeit ist die Untersuchung der BER im Kontext der Sicherheitsleistung des Systems, insbesondere im Hinblick auf die False Positive Rate (FPR) und die False Negative Rate (FNR) unter verschiedenen Testbedingungen.

Im Normalbetrieb zeigte das System eine außergewöhnlich niedrige BER, die einen maximalen Wert von 0,39% erreichte, weit unterhalb der vorgeschlagenen Toleranzgrenze von 2%. Diese Ergebnisse deuten auf eine FNR von 0% bei normalen Raumtemperaturen hin, was die hohe Zuverlässigkeit des Systems unter Standardbedingungen belegt.

Tests bei extrem niedrigen Temperaturen (ca. -17°C) zeigten jedoch in 10 von 11 Fällen einen signifikanten Anstieg der BER über die Toleranzgrenze hinaus, was zu einer FNR von fast 91% führte. Diese hohe FNR bei stark reduzierten Temperaturen weist auf eine potenzielle Schwäche des Systems unter bestimmten Umgebungsbedingungen hin und unterstreicht die Notwendigkeit, die Systemleistung über einen breiteren Bereich von Betriebsbedingungen zu optimieren. Es ist anzumerken, dass bei allen durchgeführten Tests in Angriffsszenarien die BER nie unter ca. 40% fiel, so dass es theoretisch möglich wäre, die Toleranzgrenze deutlich zu erhöhen, so dass auch diese BER-Werte nicht fälschlicherweise als manipuliert angesehen würden.

In den Angriffsszenarien, einschließlich Grounding Attack und Voltage Injection, lag die BER durchgehend in Bereichen um 50%, was weit über der Toleranzgrenze liegt. Diese Ergebnisse zeigen eine TNR von 100%, was implizit einer FPR von 0% entspricht. Dies bestätigt, dass das System tatsächlich in der Lage ist, Manipulationsversuche zu erkennen und somit die Integrität und Sicherheit der übermittelten Daten zu gewährleisten.

Die Analyse von BER, FPR und FNR liefert wertvolle Erkenntnisse über die Leistungsfähigkeit des Systems unter verschiedenen Bedingungen. Während die Ergebnisse unter normalen Bedingungen und in Angriffsszenarien die Wirksamkeit des Systems bei der Erkennung von Sicherheitsbedrohungen bestätigen, zeigen

die Tests bei niedrigen Temperaturen eine deutliche Herausforderung in Bezug auf falsch-negative Erkennungen.

6.1.4 Identifizierung und Adressierung von Schwachstellen

Ein weiteres potentielles Angriffsszenario, das während der Arbeit identifiziert wurde, ist Replay/Modification-Angriff, eine spezialisierte Form des Man-in-the-Middle-Angriffs, der die Sicherheit von Kommunikationssystemen, die dieses Authentifizierungssystem verwenden, bedrohen kann. Dieses Szenario umfasst das Einfügen eines feindlichen Mikrocontrollers in die Kommunikationskette, der in der Lage ist, Sensordaten einzulesen, die vorhandene Spannungsmodulation zu interpretieren und gefälschte Daten zurück an den eigentlichen Mikrocontroller zu senden.

Die Machbarkeit eines solchen Angriffs wurde theoretisch untersucht, wobei der Fokus auf der Fähigkeit des Angreifers lag, gefälschte Sensordaten zu erzeugen, die vom Authentifizierungssystem als authentisch, aufgrund der korrekten Spannungsmodulation, wahrgenommen werden. Obwohl die Umsetzung dieses Angriffsszenarios aufgrund bestimmter Limitationen, siehe 4.3, nicht praktisch demonstriert wurde, unterstreicht die bloße Möglichkeit eines solchen Angriffs das Bedürfnis nach weiteren Sicherheitsmaßnahmen, um die Integrität der übermittelten Sensordaten zu gewährleisten.

7 Conclusio

Die vorliegende Arbeit hat ein umfassendes Konzept zur kontinuierlichen Überprüfung von sensorseitigen Wasserzeichen entwickelt und implementiert, um die Sicherheit in Home Automation Systems und Internet-of-Things-Geräten signifikant zu verbessern. Die zentralen Forschungsergebnisse unterstreichen die Effektivität dieses Ansatzes sowohl im Normalbetrieb als auch unter spezifischen Angriffsszenarien, während gleichzeitig wichtige Einsichten in potenzielle Schwachstellen und Verbesserungsmöglichkeiten gewonnen wurden.

7.1 Hauptergebnisse

- **Hohe Systemzuverlässigkeit im Normalbetrieb:** Die Untersuchung hat eine nahezu nullprozentige BER und somit eine FNR von 0% im Normalbetrieb gezeigt, was die hohe Zuverlässigkeit des sensorseitigen Wasserzeichenkonzepts bestätigt.
- **Effektive Erkennung von Angriffen:** In Angriffsszenarien, insbesondere bei Grounding Attack und Voltage Injection, hat das System durch eine erhöhte BER alle Manipulationsversuche erfolgreich identifiziert, was eine TNR von 100% und eine FPR von 0% ergibt.
- **Herausforderungen bei extremen Bedingungen:** Tests unter extrem niedrigen Temperaturen haben jedoch eine signifikante Erhöhung der FNR gezeigt, was auf die Notwendigkeit hinweist, die Leistung und Zuverlässigkeit des Systems unter den Extrembedingungen der verwendete Sensortechnik zu untersuchen

7.2 Bedeutung und Implikationen

Die Forschungsergebnisse leisten einen wichtigen Beitrag zum Verständnis und zur Weiterentwicklung von Sicherheitsmechanismen in HAS und IoT-Geräten. Die Implementierung einer kontinuierlichen Verifikation mittels sensorseitiger Wasserzeichen bietet einen vielversprechenden Ansatz zur Erhöhung der Systemresilienz gegenüber potentiellen Angriffen. Die Identifizierung spezifischer Schwachstellen und die Reaktion

darauf eröffnen neue Wege zur Verbesserung der Sicherheitsarchitektur dieser Systeme.

7.3 Limitierungen

Die vorliegende Arbeit hat auch die Grenzen des derzeitigen Ansatzes aufgezeigt, insbesondere in Bezug auf Umgebungsbedingungen wie extreme Temperaturen. Zukünftige Forschung sollte sich auf die Optimierung der Systemleistung über ein breites Spektrum von Betriebsbedingungen konzentrieren und insbesondere Methoden zur Reduzierung der FNR unter extremen Bedingungen untersuchen.

Ein stark limitierender Faktor bei der Umsetzung des Konzepts war die Verwendung des ESP32, der trotz seiner einfachen Bedienbarkeit und schnellen Prozessleistung aufgrund seiner schlechten DAC-Auflösung sowie der extremen Nichtlinearität des ADCs Probleme bei der Umsetzung und Genauigkeit der Detektion verursachte.

7.4 Weiterführende Arbeiten

Die in dieser Arbeit identifizierten Einschränkungen und die daraus abgeleiteten Empfehlungen zeigen verschiedene vielversprechende Richtungen für zukünftige Forschungsarbeiten im Bereich sensorseitiger Wasserzeichen für die kontinuierliche Authentifizierung auf. Insbesondere wird dringend empfohlen, die Wirksamkeit des entwickelten Konzepts unter Verwendung verschiedener Hardwarekomponenten und in Extremsituationen zu untersuchen. Solche Untersuchungen würden nicht nur die Flexibilität und Übertragbarkeit des Ansatzes demonstrieren, sondern auch seine Robustheit und Zuverlässigkeit unter variierenden und potenziell herausfordernden Bedingungen unterstreichen.

Ein weiterer wichtiger Forschungsbereich ist die Entwicklung eines umfassenden Sicherheitsframeworks, das sensorseitige Wasserzeichen mit anderen Sicherheitsmechanismen kombiniert. Ein solcher Rahmen könnte einen mehrschichtigen Sicherheitsansatz bieten, der eine robuste Verteidigung gegen eine Vielzahl von Bedrohungen ermöglicht.

Abbildungsverzeichnis

List of Listings

Literatur

[1] Henri Ruotsalainen, Albert Treytl und Thilo Sauter, "Watermarking Based Sensor Attack Detection in Home Automation Systems", in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, S. 1–8. DOI: 10.1109/ETFA45728.2021.9613634.

[2] Pardeep Kumar, Andrei Gurtov, Jari Iinatti, Mika Ylianttila und Mangal Sain, "Lightweight and Secure Session-Key Establishment Scheme in Smart Home Environments", *IEEE Sensors Journal*, Jg. 16, Nr. 1, S. 254–264, 2016. DOI: 10.1109/JSEN.2015.2475298.

[3] Vandana Dhiman und Padmavati Khandnor, "Watermarking schemes for secure data aggregation in wireless sensor networks: A review paper", in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, S. 3093–3098. DOI: 10.1109/ICEEOT.2016.7755270.

[4] Anomadarshi Barua und Mohammad Abdullah Al Faruque, "Special Session: Noninvasive Sensor-Spoofing Attacks on Embedded and Cyber-Physical Systems", in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, S. 45–48. DOI: 10.1109/ICCD50377.2020.00024.

[5] Yanpeng Guan und Xiaohua Ge, "Distributed Attack Detection and Secure Estimation of Networked Cyber-Physical Systems Against False Data Injection Attacks and Jamming Attacks", *IEEE Transactions on Signal and Information Processing over Networks*, Jg. 4, Nr. 1, S. 48–59, 2018. DOI: 10.1109/TSIPN.2017.2749959.

[6] Kang Yang, Rui Wang, Yu Jiang, Chenxia Luo, Yong Guan, Xiaojuan Li und Zhiping Shi, "Enhanced Resilient Sensor Attack Detection Using Fusion Interval and Measurement History", in *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2018, S. 1–3. DOI: 10.1109/CODESISSS.2018.8525941.

[7] Vandana Dhiman und Padmavati Khandnor, "Watermarking schemes for secure data aggregation in wireless sensor networks: A review paper", in *2016 International Conference on Electrical, Elec-*

tronics, and Optimization Techniques (ICEEOT), 2016, S. 3093–3098. DOI: 10.1109/ICEEOT.2016.7755270.

[8] Thilo Sauter und Albert Treytl, "IoT-Enabled Sensors in Automation Systems and Their Security Challenges", *IEEE Sensors Letters*, Jg. 7, Nr. 12, S. 1–4, 2023. DOI: 10.1109/LSENS.2023.3332404.

[9] Yilin Mo, Sean Weerakkody und Bruno Sinopoli, "Physical Authentication of Control Systems: Designing Watermarked Control Inputs to Detect Counterfeit Sensor Outputs", *IEEE Control Systems Magazine*, Jg. 35, Nr. 1, S. 93–109, 2015. DOI: 10.1109/MCS.2014.2364724.

[10] TASMOTA, *ESP32 Based Devices*, Accessed: 04.04.2024, 2024. Adresse: https://templates.blakadder.com/esp32.html.

[11] *ESP32-WROOM-32 Datasheet*, Version 3.4, Espressif Systems, Feb. 2023.

[12] Yu Cong, Zhou Wang-chao, Sun Bin und Zhou Hang-xia, "Study on NTC thermistor characteristic curve fitting methods", in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, Bd. 4, 2011, S. 2209–2213. DOI: 10.1109/ICCSNT.2011.6182415.

[13] Jongwon Kim und Jong Dae Kim, "Voltage divider resistance for high-resolution of the thermistor temperature measurement", *Measurement*, Jg. 44, Nr. 10, S. 2054–2059, 2011, ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2011.08.004. Adresse: https://www.sciencedirect.com/science/article/pii/S0263224111002594.

[14] Chan-Young Park, Jong-Dae Kim, Ji-Min Kim, Yu-Seop Kim, Hye-Jeong Song und Jongwon Kim, "Buffer-less system for thermistor temperature measurement", in *2012 International Conference on ICT Convergence (ICTC)*, 2012, S. 240–242. DOI: 10.1109/ICTC.2012.6386828.

[15] QTI Sensing Solutions Engineering Department, "An Explanation of the Beta and Steinhart-Hart Equations for Representing the Resistance vs. Temperature Relationship in NTC Thermistor Materials", QTI Sensing Solutions, Techn. Ber., White Paper.

[16] *MCP413X/415X/423X/425X Datasheet*, DS22060B, Version 3.4, Microchip Technology Inc., Feb. 2008.

[17] *Datasheet Raspberry Pi 4 Model B*, Release 1.1, Raspberry Pi Ltd., März 2024.

[18] Arduino, *Arduino Documentation*, Accessed: 01.04.2024, 2024. Adresse: https://docs.arduino.cc/.

[19] Links2004, *Arduino WebSockets*, Accessed: 04.04.2024, 2023. Adresse: https://github.com/Links2004/arduinoWebSockets.

[20] Benoit Blanchon, *ArduinoJson*, Accessed: 04.04.2024, 2023. Adresse: https://arduinojson.org/.

[21] Raspberry Pi Ltd., *Raspberry Pi Documentation*, Accessed: 04.04.2024, 2024. Adresse: https://www.raspberrypi.com/documentation/.

[22] Python Software Foundation, *asyncio — Asynchronous I/O*, Accessed: 04.04.2024, 2024. Adresse: https://docs.python.org/3/library/asyncio.html.

[23] Andrew Svetlov and Nikolay Kim, *aiohttp*, Accessed: 04.04.2024, 2024. Adresse: https://docs.aiohttp.org/.

[24] Aymeric Augustin and Mitwirkende, *websockets*, Accessed: 04.04.2024, 2024. Adresse: https://websockets.readthedocs.io/.

[25] NumPy, *NumPy Documentation*, Accessed: 04.04.2024, 2024. Adresse: https://numpy.org/doc/.

[26] Python Software Foundation, *json — JSON encoder and decoder*, Accessed: 04.04.2024, 2024. Adresse: https://docs.python.org/3/library/json.html.

[27] Mike Bayer, *SQLAlchemy 2.0 Documentation*, Accessed: 04.04.2024, 2024. Adresse: https://docs.sqlalchemy.org/.

[28] uPesy, *How to connect to a WiFi network with the ESP32*, Accessed: 31.03.2024, Jan. 2023. Adresse: https://www.upesy.com/blogs/tutorials/how-to-connect-wifi-acces-point-with-esp32.

[29] Isa Anshori, Ghani Mufiddin, Iqbal Ramadhan, Eduardus Ariasena, Suksmandhira Harimurti, Henke Yunkins und Cepi Kurniawan, "Design of smartphone-controlled low-cost potentiostat for cyclic voltammetry analysis based on ESP32 microcontroller", *Sensing and Bio-Sensing Research*, Jg. 36, S. 100 490, März 2022. DOI: 10.1016/j.sbsr.2022.100490.

[30] Philip Kane, *Temperature Measurement with NTC Thermistors*, Accessed: 04.04.2024, 2024. Adresse: https://www.jameco.com/Jameco/workshop/TechTip/temperature-measurement-ntc-thermistors.html.

[31] Alam Rahmatulloh, Irfan Darmawan und Rohmat Gunawan, “Performance Analysis of Data Transmission on WebSocket for Real-time Communication”, in *2019 16th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*, 2019, S. 1–5. DOI: 10.1109/QIR.2019.8898135.

[32] Chong Hee Kim und Jean-Jacques Quisquater, “Faults, Injection Methods, and Fault Attacks”, *IEEE Design Test of Computers*, Jg. 24, Nr. 6, S. 544–545, 2007. DOI: 10.1109/MDT.2007.186.

[33] D. Page und F. Vercauteren, “A Fault Attack on Pairing-Based Cryptography”, *IEEE Transactions on Computers*, Jg. 55, Nr. 9, S. 1075–1080, 2006. DOI: 10.1109/TC.2006.134.

[34] Fang Xupeng, Zheng Shicheng, Li Chenna, Wang Qingqing, Qi Zhongming und Wang Pu, “Analysis of Short Circuit Fault Detection of Power Switch of Impedance-source Inverter Based on Shoot-through Characteristic”, in *2020 Chinese Automation Congress (CAC)*, 2020, S. 5290–5295. DOI: 10.1109/CAC51589.2020.9326466.

[35] Duko Karaklaji, Jörn-Marc Schmidt und Ingrid Verbauwhede, “Hardware Designer’s Guide to Fault Attacks”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Jg. 21, Nr. 12, S. 2295–2306, 2013. DOI: 10.1109/TVLSI.2012.2231707.

[36] B M Ruhul Amin, M. S. Rahman und M. J. Hossain, “Impact Assessment of Credible Contingency and Cyber Attack on Australian 14-Generator Interconnected Power System”, in *2019 IEEE Power Energy Society General Meeting (PESGM)*, 2019, S. 1–5. DOI: 10.1109/PESGM40551.2019.8973851.

[37] Noemie Beringuier-Boher, Kamil Gomina, David Hely, Jean-Baptiste Rigaud, Vincent Beroulle, Assia Tria, Joel Damiens, Philippe Gendrier und Philippe Candelier, “Voltage Glitch Attacks on Mixed-Signal Systems”, in *2014 17th Euromicro Conference on Digital System Design*, 2014, S. 379–386. DOI: 10.1109/DSD.2014.14.

[38] Otto Bittner, Thilo Krachenfels, Andreas Galauner und Jean-Pierre Seifert, “The Forgotten Threat of Voltage Glitching: A Case Study on Nvidia Tegra X2 SoCs”, in *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, 2021, S. 86–97. DOI: 10.1109/FDTC53659.2021.00021.

[39] M. Andjelkovic, R. T. Syed, M. Pavlovic, F. Vargas, T. Nikolic, G. Ristic und M. Krstic, “Voltage Glitch Filter and Detector with Self-Checking Capability for FPGA Implementation”, in *2023 IEEE 33rd International Conference on Microelectronics (MIEL)*, 2023, S. 1–4. DOI: 10.1109/MIEL58498.2023.10315811.

[40] Thomas Roth, Fabian Freyer, Matthias Hollick und Jiska Classen, "AirTag of the Clones: Shenanigans with Liberated Item Finders", in *2022 IEEE Security and Privacy Workshops (SPW)*, 2022, S. 301–311. DOI: 10.1109/SPW54247.2022.9833881.

[41] LimitedResults, *NRF52 debug resurrection (approtect bypass) part 1*, Accessed: 31.03.2024, Mai 2021. Adresse: https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass/.