Chair for Embedded Systems
Universität Augsburg

Master's Thesis

Implementation of an IoT based Electronic Voting Machine

Gabriel Cmiel

Gutachter/Examiner:	Prof. Dr. Vorname Nachname
Zweitgutachter/Second examiner:	Prof. Dr. Vorname Nachname
Betreuer/Supervisor:	Prof. Dr. Sebastian Altmeyer
Date:	16th October 2024

written at
Chair for Embedded Systems
Prof. Dr. Sebastian Altmeyer
Institute of Computer Science
University of Augsburg
D–86135 Augsburg, Germany
https://www.Informatik.uni-augsburg.de

Contents

Abstract

Eine kurze Zusammenfassung der Ausarbeitung.

Nomenclature

EAC Election Assitance Commission

E2E End-to-end

VVSG Voluntary Voting System Guidelines

1. Einleitung/Introduction

In der Einleitung wird die Arbeit motiviert und die Relevanz dieser herausgearbeitet.

1.1. Ziele dieser Arbeit

Die Ziele der Arbeit werden hier erläutert.

1.2. Überblick

Der Autor führt einen potentiellen Leser durch die Arbeit und beschreibt kurz, was den Leser in den folgenden Kapiteln erwartet.

2. Grundlagen/Fundamentals

2.1. Cryptography

Cryptography is the science of securing information through encryption. Encryption or ciphering refers to the process of making a message incomprehensible **crypto** The security of all cryptographic methods is essentially based on the difficulty of guessing a secret key or obtaining it by other means. It is possible to guess a key, even if the probability becomes very small as the length of the key increases. It must be pointed out that there is no absolute security in cryptography **crypto**.

Practically all cryptographic methods have the task of ensuring one of the following security properties are met **crypto**.

- **Confidentiality** The aim of confidentiality is to make it impossible or difficult for unathorized persons to read a message **crypto**.

- **Authenticity** Proof of identity of the message sender to the recipient, i.e. the recipient can be sure that the message does not originate from another (unauthorized) sender **crypto**

- **Integrity** The message must not be altered (by unauthorized persons) during transmission. It retains its integrity **crypto**.

- **Non-repudiation** The sender cannot later deny having sent a message **crypto**.

Cryptographic algorithms are mathematical equations, i.e. mathematical functions for encryption and decryption **crypto**. A cryptographic algorithm for encryption can be used in a variety of ways in different applications. To ensure that an application always runs in the same and correct correct way, cryptographic protocols are defined. In contrast to the cryptographic algorithms, the protocols are procedures for controlling the flow of transactions for certain applications. **crypto**.

The idea of combining cryptographic methods with voting systems is not new. In 1981, David Chaum published a cryptographic technique based on public key cryp-

tography that hides who a participant communicates with, aswell as the content of the communication. The untracable mail system requires messages to pass through a cascade of mixes (also known as a Mix Network) **chaum**. Chaum proposes that the techniques can be used in elections in which an individual can correspond with a record-keeping organisation or an interested party under a unique pseudonym. The unique pseudonym has to appear in a roster of accetable clients. A interested party or record keeping organisation can verify that the message was sent by a registered voter. The record-keeping organisation or the interested party can also verify that the message was not altered during transmission. **chaum**.

In this use case, the properties of Confidentiality, Authenticity, Integrity and Non-repudiation are ensured. However, to be worthy of public trust, an election process must give voters and observers compelling evidence that the election was conducted properly without breaking voters confidentiality. The problem of public trust is further exacerbated to now having to trust election software and hardware, in addition to election officials, and proceedurs.

In 2021, the U.S. Election Assitance Commission (EAC) adopted the Voluntary Voting System Guidelines (VVSG) 2.0. **eac-pressrelease**. The VVSG is intended for designers and manufacturers of voting systems. Currently, the VVSG is titled as "Recommendations to the EAC" because it's not yet the final version that voting system manufacturers will follow. **https://www.nist.gov/itl/voting/vvsg-introduction**. The VVSG 2.0 currently states only two methods for achieving software independence. The first through the use of independent voter-verifiable paper records, and the second through cryptographic E2E verifiable voting systems. The VVSG 2.0 states that a voting system need to be software independent through the use of independent voter-verifiable paper records **https://www.eac.gov/sites/default/files/TestingCertificatio**
However, due to the lack of E2E verifiable voting systems available within the current market, there are no verified E2E cryptographic protocols.**https://www.eac.gov/sites/default/**
The U.S. Election Assistance Commission, in collaboration with the National Institute of Standards and Technology initialised an Call for proposals to solicit, evaluate, and approve protocols used in E2E cryptographically verifiable voting systems. **https://www.eac.gov/voting-equipment/end-end-e2e-protocol-evaluation-process**.

Submitted protocols must support the following properties

- Cast as Intended: Allow voters to confirm the voting system correctly interpreted their ballot selections while in the polling place via a receipt and provide evidence such that if there is an error or flaw in the interpretation of the voters' selections.

- Recorded as Cast: Allow voters to verify that their cast ballots were accurately recorded by the voting system and included in the public records of encoded ballots.

- Tallied as Recorded: Provide a publicly verifiable tabulation process from the public records of encoded ballots.

2.2. ElectionGuard

One of the first pilots to see how E2E verifiable elections works in a real election took place in a district of Preston, Idaho, United States, on November 8, 2022. The Verity scanner from Hart InterCivic was used in this pilot, which was integrated with Microsoft's ElectionGuard. [EAC Report]. ElectionGuard is a toolkit that encapsulates cryptographic functionality and provides simple interfaces that can be used without cryptographic expertise. The principal innovation of ElectionGuard is the seperation of the cryptographic tools from the core mechanics and user interfaces of voting systems. In it's preferred deployment, ElectionGuard doesn't replace the existing vote counting infrastructure but instead runs alongside and produces its own independently-verifiable tallies **eg-paper**. The cryptographic design is largely inspired by the cryptographic voting protocol by Cohen (now Benaloh) and Fischer in 1985 and the voting protocol by Cramer, Gennaro and Schoenmakers in 1997 **eg-paper**.

———————————————- The philosophy of ElectionGuard has been to cover the majority of voting scenarios with an approach that is as simple as possible to understand and verify. It can be used with Precinct Ballot Scanners, Electronic Ballot Markers, Internet Voting, Risk-Limiting Audits and even vote by mail and many more.

In all applications, an election using ElectionGuard begins with a key-generation ceremony in which an election administrator works with guardians to form election keys. Later, usually at the conclusion, the administrator will again work with guardians to produce verifiable tallies. What happens in between, however, can vary widely. [ElectionGuard: a Cryptographic Toolkit to Enable Verifiable Elections].

This thesis focuses on the implementation of the Key Generation Ceremony and the Guardians using the ESP32 microcontroller. The ESP32 is a low-cost, low-power system on a chip microcontroller. It is widely used in IoT applications. [source]
———————————————-

2.2.1. Election Verification

ElectionGuard support the central aspects which a VVSG 2.0 compliant voting system must support Cast as Intended, Recorded as Cast and Tallied as Recorded. **eg-paper**. Key element supporting cast-as-intended and recorded-as-cast verifiability is through confirmation codes. Ballots can be ce challenged or cast and both are included in the election record. Voters can check if the expected confirmation code appears in the election records and for the challenged ballots, that it shows the correct selections **eg-paper**. Tallied-as-cast verifiability is supported through the inclusion of all ballots and decryption proofs in the election record. Any voter can verify that the ballot is accurately incorporated in the tally, and the decryption proofs demonstrate the validity of the announced tally **eg-paper**. To confirm the election's integrity, independent verification software can be used at any time after the completion of an election. **eg-paper**. Some, may choose even to write their own verifiers.

One such E2E verifiable voting protocol is ElectionGuard which will be implemented in this thesis. Before proceeding several cryptographic techniques need to be introduced first.

End-to-end verifiable election techniques enable individual voters to check crucial ingredients of election results – without requiring voters to trust election software, hardware, election officials, procedures, or even observers. Voters may check these ingredients themselves, place their trust in others of their choice (e.g. their preferred candidates, news media, and/or interest groups), or accept the outcome produced with the usual administrative safeguards page1

Any chain of evidence, for any system design, requires some assumptions. Checking that those assumptions hold for the election in question is crucial. Trusting unverifiable hardware or software is generally not reasonable. Trusting processes for secure handling of paper ballots may, or may not, be reasonable, depending on what those processes are and how they can be observed. E2E-V systems are software independent and do not rely on trusted paper processes, instead providing opportunities to verify the election outcome electronically page2

End-to-end verifiable voting schemes provide ways of performing these tasks.

ElectionGuard's approach to End-to-End Verifiable Elections

Core Principles of ElectionGuard Guardian key and tally ceremonies Generation of ballot confirmation code Cofnirmation-code lookup Seite Ability to challenge ballots Verifier support

Integrating ElectionGuard's key ceremony with ESP32

2.3. ElectionGuard

ElectionGuard is not a complete election system. It instead provides components for system developers to implement E2E-verifiable elections. The goal of ElectionGuard is to promote voter confidence by empowering voters to indepently verify the accuracy of election results. **eg-spec**

ElectionGuard achieves this by

2.3.1. Verifier Construction

the signature. The entire election record and its digital signature should be **eg-spec** published and made available for full download by any interested individuals. Tools should also be provided for easy look up of tracking codes by voters. **eg-spec**

2.3.2. ElectionGuard Components

Four Principal components of ElectionGuard are described below **eg-spec**

neter Requirements These are properties required of parameters that are standard in every election. A specific set of standard parameters is provided [...].

Key Generation Prior to each individual election, guardians must generate individual publicprivate key pairs and exchange shares of private keys to enable completion of an election even if some guardians become unavailable.

Ballot Encryption While encrypting the contents of a ballot is a relatively simple operation, most of the work of ElectionGuard is the process of creating externally-verifiable artifacts to prove that each encrypted ballot is well-formed (i.e., its decryption is a legitimate ballot without overvotes or improper values).

erifiable Decryption At the conclusion of each election, guardians use their private keys to produce election tallies together with verifiable artifacts that prove that the tallies are correct.

Parameter Requirements

Ballot Encryption

Ballot encryption In most uses, the election system makes a single call to the ElectionGuard API after each voter completes the process of making selections [...]. ElectionGuard will encrypt the selections made by the voter and return a verification code which the system should give to the voter.1 **eg-spec**

The encrypted ballots are published along with non-interactive zero-knowledge proofs of their integrity. The encryption method used herein has a homomorphic property which allows the encrypted ballots to be combined into a single aggregate ballot which consists of encryptions of the election tallies. **eg-spec**

Encryption of votes in ElectionGuard is performed using an exponential form of the ElGamal cryptosystem4. **eg-spec**.

An encryption of one is used to indicate that an option is selected, and an encryption of zero is used to indicate that an option is not selected.**eg-spec**

Homomorphic properties A fundamental quality of the exponential form of ElGamal described above is its additively homomorphic property. If two messages M1 and M2 are respectively encrypted as $(A1, B1) = (gR1 mod p, gM1 \cdot KR1 mod p) and (A2, B2) = (gR2 mod p, gM2 \cdot KR2 mod p)$,then the componentwise product $(A, B) = (A1A2 mod p, B1B2 mod p) = (gR1 + R2 mod p, gM1 + M2 \cdot KR1 + R2 mod p)$ is an encryption of the sum $M1 + M2$. **eg-spec**.

This additively homomorphic property is used in two important ways in ElectionGuard. First, all of the encryptions of a single option across ballots can be multiplied to form an encryption of the sum of the individual values. Since the individual values are one on ballots that select that option and zero otherwise, the sum is the tally of votes for that option and the product of the individual encryptions is an encryption of the tally. The other use is to sum all of the selections made in a single contest on a single ballot. After demonstrating that each option is an encryption of either zero or one, the product of the encryptions indicates the number of options that are encryptions of one, and this can be used to show that no more ones than permitted are among the encrypted options – i.e., that no more options were selected than permitted. However, as will be described below, it is possible for a holder of a nonce R to prove to a third party that a pair (α, β) is an encryption of M without revealing the nonce R and without access to the secret s. **eg-spec**

Encryption of other data ElectionGuard provides means to encrypt data other than

votes, which are selections encoded as zero or one that need to be homomorphically aggregated. Such data may include the cryptographic shares of a guardian's private key that are sent to the other guardians and are encrypted to the receiving guardians' public keys, write-in information that needs to be attached to an encrypted selection and is encrypted to the election public key, or other auxiliary data attached to an encrypted ballot, either encrypted to the election public key or to an administrative public key. The non-vote data do not need to be homomorphically encrypted and can use a more standard form of public-key encryption removing the data size restrictions imposed by the exponential ElGamal scheme for vote encryption. ElectionGuard encrypts such data with hashed ElGamal encryption, which deploys a key derivation function (KDF) to generate a key stream that is then XORed with the data. To implement the KDF and to provide a message authentication code (MAC), encryption makes use of the keyed Hash Message Authentication Code HMAC. In ElectionGuard, HMAC is instantiated as HMAC-SHA-256 with the hash function SHA-256. **eg-spec**

Verifiable Decryption

After the conclusion of voting or auditing, a quorum of guardians is necessary to produce the artifacts required to enable public verification of the tally. **eg-spec**

Verifiable decryption In the final step, election guardians independently use their secret keys to decrypt the election tallies and associated verification data. It is not necessary for all guardians to be available to complete this step. If some guardians are missing, a quorum of guardians can use the previously shared key fragments to reconstruct the missing verification data. **eg-spec**

Observers can use this open specification and/or accompanying materials to write election verifiers that can confirm the integrity of each encrypted ballot, the correct aggregation of these ballots, and the accurate decryption of election tallies. **eg-spec**

Threshold encryption Threshold ElGamal encryption is used for encryption of ballots and other data. This form of encryption makes it very easy to combine individual guardian public keys into a single public key. It also offers a homomorphic property that allows individual encrypted votes to be combined to form encrypted tallies. The guardians of an election will each generate a public-private key pair. The public keys will then be combined (as described in the following section) into a single election public key which is used to encrypt all selections made by voters in the election. Ideally, at the conclusion of the election, each guardian will use its private key to form a verifiable partial decryption of each tally. These partial decryptions will then be combined to form full verifiable decryptions of the election

tallies. To accommodate the possibility that one or more of the guardians will not be available at the conclusion of the election to form their partial decryptions, the guardians will **eg-spec**

2.3.3. Parameter Requirements

share11 their private keys amongst each other during key generation in a manner to be detailed in the next section. A pre-determined threshold quorum value (k) out of the (n) guardians will be necessary to produce a full decryption. **eg-spec**

Parameter Requirements Specific values for primes p and q and a generator g are given in Section 4 Baseline Parameters below. **eg-spec**

Note that decryption of individual ballots does not directly compromise voter privacy since links between encrypted ballots and the voters who cast them are not retained by the system. However, voters receive verification codes that can be associated with individual encrypted ballots, so any group that has the ability to decrypt individual ballots can also coerce voters by demanding to see their tracking codes. Threshold ElGamal encryption is used for encryption of ballots. This form of encryption makes it very easy to combine individual guardian public keys into a single public key for encrypting votes and ballots. It also offers a homomorphic property that allows individual encrypted votes to be combined to form encrypted tallies. The guardians of an election will each generate a public-private key pair. The public keys will then be combined (as described in the following section) into a single election public key which is used to encrypt all selections made by voters in the election. Ideally, at the conclusion of the election, each guardian will use its private key to form a verifiable partial decryption of each tally. These partial decryptions will then be combined to form full verifiable decryptions of the election tallies. To accommodate the possibility that one or more of the guardians will not be available at the conclusion of the election to form their partial decryptions, the guardians will cryptographically share13 their private keys amongst each other during key generation in a manner to be detailed in the next section. A pre-determined threshold value (k) out of the (n) guardians will be necessary to produce a full decryption. **eg-spec**

2.3.4. Ballot Encryption

An ElectionGuard ballot is comprised entirely of encryptions of one (indicating selection made) and zero (indicating selection not made). To enable homomorphic addition (for tallying), these values are exponentiated during encryption. **eg-spec**

A contest in an election consists of a set of options together with a selection limit that indicates the number of selections that are allowed to be made in that contest. In most elections, most contests have a selection limit **eg-spec**

A legitimate vote in a contest consists of a set of selections with cardinality not exceeding the selection limit of that contest. To accommodate legitimate undervotes, the internal representation of a contest is augmented with "placeholder" options equal in number to the selection limit. Placeholder options are selected as necessary to force the total number of selections made in a contest to be equal to the selection limit. When the selection limit is one, for example, the single placeholder option can be thought of as a "none of the above" option. With larger selection limits, the number of placeholder options selected corresponds to the number of additional options that a voter could have selected **eg-spec** options selected corresponds to the number of additional options that a voter could have selected in a contest. For efficiency, the placeholder options could be eliminated in an approval vote. However, to **eg-spec**

Two things must now be proven about the encryption of each vote to ensure ballot integrity. 1. The encryption associated with each option is either an encryption of zero or an encryption of one. 2. The sum of all encrypted values in each contest is equal to the selection limit for that contest (usually one). **eg-spec**

2.3.5. Outline for proofs of ballot correctness

The use of ElGamal encryption enables efficient zero-knowledge proofs of these requirements, and the Fiat-Shamir heuristic can be used to make these proofs non-interactive. Chaum-Pedersen proofs are used to demonstrate that an encryption is that of a specified value, and these are combined with the Cramer-Damg ard-Schoenmakers technique to show that an encryption is that of one of a specified set of values – particularly that a value is an encryption of either zero or one. The encryptions of selections in a contest are homomorphically combined, and the result is shown to be an encryption of that contest's selection limit – again using a Chaum-Pedersen **eg-spec** the result is shown to be an encryption of that contest's selection limit – again using a Chaum-Pedersen proof. **eg-spec** The "random" nonces used for the ElGamal encryption of the ballot nonces are derived from a single 256-bit master nonce RB for each ballot. **eg-spec**

2.3.6. Tracking codes

Upon completion of the encryption of each ballot Bi, a tracking code Hi is prepared for each voter. The code is a SHA-256 hash **eg-spec** previous tracking code. When the previous tracking code Hi is included, the chain is The benefit of a chain is that it makes it more difficult for a malicious insider to selectively delete ballots and tracking codes after an election without detection. **eg-spec**

Once in possession of a tracking code (and never before), a voter is afforded an option to either cast the associated ballot or spoil it and restart the ballot preparation process. The precise mechanism for voters to make these selections may vary depending upon the instantiation, but this choice would ordinarily be made immediately after a voter is presented with the tracking code, and the status of the ballot would be undetermined until the decision is made. It is possible, for instance, for a voter to make the decision directly on the voting device, or a voter may instead be afforded an option to deposit the ballot in a receptacle **eg-spec**

2.3.7. Ballot Aggregation

At the conclusion of voting, all of the ballot encryptions are published in the election record together with the proofs that the ballots are well-formed. Additionally, all of the encryptions of each option are homomorphically combined to form an encryption of the total number of times that option was selected. The encryptions $(\alpha i, \beta i)$ of each individual option are combined by forming the product $(A, B) = (i\alpha imodp, i\beta imodp)$. This aggregate encryption (A, B), which represents an encryption of the tally of that option, is published in the election record for each option. **eg-spec**

2.3.8. Decryption of spoiled ballots

Every ballot spoiled in an election is individually verifiably decrypted in exactly the same way that the aggregate ballot of tallies is decrypted. **eg-spec**

The exponential ElGamal used to encrypt votes is defined by a 4096-bit prime p and a 256-bit prime q which divides (p - 1). We use r = (p - 1)/q to denote the cofactor of q, and a **eg-spec**

2.3.9. The Election Record

The record of an election should be a full accounting of all of the election artifacts. Specifically, it should contain the following. Date and location of an election. The ballot coding file. The baseline parameters: **eg-spec**

Every encrypted ballot prepared in the election (whether cast or spoiled): – All of the encrypted options on each ballot (including "placeholder" options), – the proofs that each such value is an encryption of either zero or one, – the selection limit for each contest, – the proof that the number of selections made matches the selection limit, – the device information for the device that encrypted the ballot, – the date and time of the ballot encryption, – the tracker code produced for the ballot. The decryption of each spoiled ballot: – The selections made on the ballot, – the cleartext representation of the selections, – partial decryptions by each guardian of each option, – proofs of each partial decryption, – shares of each missing partial decryption (if any), – proofs of shares of each missing partial decryption (if any), – Lagrange coefficients used for replacement of any missing partial decryptions (if any). Tallies **eg-spec**

Lagrange coefficients used for replacement of any missing partial decryptions (if any). Tallies of each option in an election: – The encrypted tally of each option, – full decryptions of each encrypted tally, – cleartext representations of each tally, – partial decryptions by each guardian of each tally, – proofs of partial decryption of each tally, – shares of each missing partial decryption (if any), – proofs of shares of each missing partial decryption (if any), – Lagrange coefficients used for replacement of any missing partial decryptions (if any). Ordered lists of the ballots encrypted by each device. An election record should be digitally signed by election administrators together with the date of 24 **eg-spec**

2.4. KiCad EDA

3. Hauptteil/Main Part

3.1. Hardware

3.1.1. Breadboard Prototype

Our prototype uses the NodeMCU ESP32 development board by Joy-IT. The board is equipped with the ESP32-WROOM-32 module. ESP32-WROOM-32 is a powerful, generic Wi-Fi+BT+BLE MCU module. The module

At the core of the module is the ESP32-D0WDQ6 chip **esp32-module**. This chip, and therefore this module. is not recommended for new designs anymore due to chip revisions. The ESP32-D0WDQ6 is based on chip revision v.1.0 or v1.1. **esp32-datasheet**. The ESP32-WROOM-32 module could therefore be replaced by the newer ESP32-WROOM-32E module. The ESP32-WROOM-32E module uses a ESP32-D0WD-V3 or ESP32-D0WDR2-V3 chip which are based on chip revision v3.0 or v3.1 which fixes some Hardware bugs **esp32-module-new**, **esp32-datasheet**, **esp32-errata**. Compared to the ESP32-WROOM-32 module, the ESP32-WROOM-32E module has versions that provide additional 2MB PSRAM, support higher operating ambient temperatures and versions with higher integrated SPI flash sizes (8MB or 16MB) **esp32-module-new**, **esp32-module**.

3.1.2. PCB Design

Schematic

Design

BOM

3.2. Implementation/Software Architecture

3.2.1. Model

Difference with other Implementations

Since ElectionGuards original specification in 2019, there have been several implementations of ElectionGuard that have been used in various applications . The current roadmap of ElectionGuard targets a c++ implementation of the ElectionGuard 2.0 specification. [https://www.electionguard.vote/overview/Roadmap/]. Earlier implementations include a Python reference implementation of the ElectionGuard 1.0 specification and an encryption engine in C++ with a C wrapper.

ESP32 development support development of applications in C, C++ and Micropython. [source]. Initialy, we could try to port the encryption engine written in c++ over to ESP32. This would allow us to use the existing codebase and focus on the integration of the encryption component with the hardware. The encryption engine would be responsible for generating an elgamal keypair and the subsequent exchange of cryptographic proofs and cryptographic keys.

The modular exponentiation at the heart of most ElectionGuard operations imposes the highest computaitional cost among all computations and is the limiting factor in any performance analysis. Using fast libraries for modular arithmetic is crucial to achieve good performance so that the latency due to Key generation and ZK proof generation doesn not impact usability. [source]

The c++ implementation uses Microsoft's HACL* - a performant C implementation of a wide variety of cryptographic primitives which have been formally verified for correctness [source.]. For performance reasons the implementation of the c++ encryption engine uses pre-computed tables to make encryption substantially faster. This is possible because most exponentiations in ElectionGuard have fixed base,

either the generator g or the election public key K. The pre-computed tables contain certain powers of these bases. The Python reference implementation uses a more straightforwar approach by using GnuMP. [source].

The ESP32 is equipped with hardware accelerators of general algorithms such as SHA and RSA and it also supports independent arithmetic, such as Big Integer Multiplication and Big Integer Modular Multiplication. [esp tech reference,4.1.19]. The hardware accelerators greatly improve operation speed and reduce software complexity.

Performance

- With/Without HW Accerleration - Single/Dual Core

3.2.2. View

3.2.3. Adapter

3.3. Verification

	LCD	Board	Description
1	VCC	3.3V	
2	GND	GND	
3	GND	GND	
4	NC		
5	NC		
6	NC		
7	CLK	D14	SPI-CLK
8	SDA	D13	SPI-MOSI
9	RS	D34	Any GPIO PIN
10	RST	D35	Any GPIO PIN
11	CS	D15	SPI-SS

Table 3.1.: PINOUT LCD

Bachelor- und Masterarbeiten können sowohl in deutsch als auch in englisch geschrieben werden. Die sprachliche Ausarbeitung wird bewertet, was bei der Wahl

der Sprache berücksichtigt werden sollte. Im folgenden werden ein paar Hinweise zur Ausarbeitung mit LaTeX gegeben.

3.4. Unterkapitel

3.4.1. Dritte Gliederungsebene

Falls in einem Kapitel mehrere Gliederungsebenen verwendet werden sollte darauf geachtet werden, dass mindestens drei Punkte pro ebene existieren.

1	2	3
4	5	6

Table 3.2.: Beispieltabelle

Hier wird die Beispieltabelle 3.2 referenziert.

Figure 3.1.: Bildunterschrift mit Quellenangabe **lcd**

Hier wird die Beispielbild 3.1 referenziert.

$$\sum_{x=0}^{10} x = 55 \tag{3.1}$$

Hier wird die Beispielformel 3.1 referenziert.

kursiv, **fett**, <u>unterstrichen</u>

Abkürzungen müssen im Abkürzungsverzeichnis angelegt werden. Erste Verwendung einer **ABK!** (**ABK!**) jede weitere Verwendung der **ABK!**.

4. Zusammenfassung und Ausblick/Summary and Outlook

Im Schlusskapitel wird die Arbeit und ihre Ergebnisse zusammengefasst sowie ein Ausblick gegeben.

List of Figures

List of Tables

A. Appendix