CHAIR FOR EMBEDDED SYSTEMS
UNIVERSITÄT AUGSBURG

Universität
Augsburg
University

Master's Thesis

Implementation of an IoT based Electronic Voting Machine

Gabriel Cmiel

Contents

Abstract

Eine kurze Zusammenfassung der Ausarbeitung.

List of Abbreviations

EAC   Election Assitance Commission

E2E   End-to-end

VVSG   Voluntary Voting System Guidelines

ZK   zero-knowledge

PRG   Pseudo-random generator

NIZK   Non-interactive zero-knowledgte proofs

KDF   Key derivation function

MAC   Message authentication code

HMAC   Hash Message Authentication code

NIST   National Institute of Standards and Technology

SHA   Secure Hash Algorithm

MCU   Micro Controller Unit

SoC   System on a Chip

Glossary

1 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 Background

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**.

2.2 Cryptog

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

2.3 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**. 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]

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.

2.3.1 Cryptographic Design Structure

An election in electionguard comprises of Pre-election, Intra-election and Post-election phases. In the following sections, we will discuss the cryptographic design and the overall structure of ElectionGuard in each of these phases.

Pre-election

. The pre-election phase contains the administrative task of configuring the election followed by the key generation ceremony. The election is defined using an election manifest **eg-paper**. The manifest defines common elements when conducting an election, such as locations, candidates, parties, contests, and ballot styles. The election terms and the manifest structure are largely based on the NIST SP-1500-100 Election Results Common Data Format Specification and the Civics Common Standard Data Specification.**eg-docs**. The manifest guarantees that ElectionGuard software records ballots properly. **eg-paper**. Each election also has to define cryptographic parameters. One set of cryptographic parameters are the mathematical constants that will be used in the cryptographic operations. The ElectionGuard specification specifies baseline and alternative values for these mathematical constants **eg-spec**. Further cryptographic parameters are the number of guardians and the quorum count which play an important role in the key generation ceremony. **eg-paper**.

The key generation ceremony is a process in which independent and trustworthy individuals called guardians work together to generate a joint key. The joint key is created by simple multiplication of the individual public keys of the guardians.

When the joint key is used to encrypt data, the data can only be decrypted by all guardians applying their private key. A quorum count of guardians can be specified to compensate for guardians missing at the time of decryption. To compensate for missing guardians, the guardians can distribute "backups" of their private key to each other, such that a quorum of guardians can reconstruct the missing private key. **eg-paper eg-docs**.

The last pre-election step is to load the manifest, cryptographic parameters and the joint key into an encryption device. The encryption device is then used to encrypt the ballots during the election **eg-paper**.

2.3.2 Intra-election

. Encrypted ballots consist entirely of exponential ElGamal encryptions of ones and zeroes. A one indicates voter supports the choice, a zero indicates the voter does not support the corresponding choice. **eg-paper eg-spec**. If a voter has four options in a single contest, the encrypted ballot will consist of four encrypted bits. The exponential form of ElGamal has an additive homomorphic property the product of the encrypted ballot indicates the number of options that are encryptions of one. This can be used to show that the ballot does not incldue excessive votes. **eg-spec**.

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 that prove that each encryption is well-formed. **eg-spec**. In order to prove that the encryptions are encryptions of ones and zeroes, Chaum-Pedersen Proofs are used. **eg-paper**. Chaum-Pedersen proof is a zero-knowledge proof that proves to decrypt to a particular value without revealing the nonce used for encryption or the secret decryption key. **eg-spec**.

———

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 proof. **eg-spec**

——

Upon completion of the encryption of a ballot a confirmation code is prepared for the voter.**eg-spec**. The confirmation code is a cryptographic hash derived entirely from the encryption of the ballot.**eg-paper**. Once the voter is in possesion of a confirmation code, the voter can either cast the associated ballot or spoil it and restart the ballot preperation process. **eg-spec**. The casting and spoiling mechanism is an interactive proof aimed to give voters confidence that their selections have been correctly encrypted. **eg-docs**. [source].

2.3.3 Post-election

At the conclusion of voting, all encrypted ballots that are intended to be tallied i.e. submitted ballots are homomorphically combined to form an encrypted tally. **eg-spec eg-spec eg-paper**. Decrypting the individual spoiled ballots is not necessary for the election outcome. They can optionally be decrypted in order to support cast-as-intended verifiability. **eg-paper**. To decrypt an encrypted tally or a spoiled ballot each available guardian uses its secret key to compute a decryption share which is a partial decryption of each given encrypted tally or spoiled ballot **eg-spec eg-paper**. Each guardian also publishes a Chaum-Pedersen proof of the correctness of the decryption share. **eg-spec**. The partial decryptions can be combined using ordinary multiplication to form the full decryption **eg-docs**. If Guardians are missing during a decryption, the quorum of guardians can use the backups to reconstruct the missing decryption shares. **eg-docs**.

The final step of the election is to publish the election record. The value of a verifiable election is only fully realized, when the election is actually verified, for example by voters, election observers, or news organisations. **eg-spec**. The election record is a full accounting of all the election artifacts it includes items like the manifest, cryptographic parameters, decrypted tally etc. **eg-spec**. The election record is published in a public bulletin board. **eg-spec**. The election record is a full accounting of all the election artifacts.

2.4 ESP32

ESP32 is a system on a chip that integrates . . . Espressif provides software development framework.

Build System: CMake ESP-IDF package manager Compiler:GCC

3 Hauptteil/Main Part

3.1 Hardware Reference

Basic Component List	Number	Notes
NodeMCU ESP32	1	Development board for prototyping
1,8" LCD Display Module	1	128x160 Pixel, ST7735R (Display Driver), includes microSD Slot
Button module with 2 Buttons	1	For interacting
Jumper Wires Female to Male	22	For connecting components
Breadboard	1	For building circuits

Table 3.1: Basic Component List for Guardian Prototype

As shown in Table **??**, the hardware components used in our prototype include a ESP32 development board, an LCD display module, jumper wires, breadboard and a module with 2 Buttons. The connectivity of these components are shown in the schematic in Figure **??**. The NodeMCU ESP32 development board is used as the Main Controller. The LCD display module is used to display information to the user. The button module is used to interact with the user. The jumper wires are used to connect the components. The breadboard is used to build the circuits.

3.1.1 NodeMCU ESP32 Development Board

The NodeMCU ESP32 development board is equipped with the ESP32-WROOM-32 module. The ESP32-WROOM-32 module is a powerful, generic Wi-Fi+Bluetooth+Bluetooth Low Energy MCU module. At the core of this module is the System on a Chip (SoC)ESP32-D0WDQ6. **esp32-module**. The ESP32-D0WDQ6 chip is based on chip revision v.1.0 or v1.1. **esp32-ds**. 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
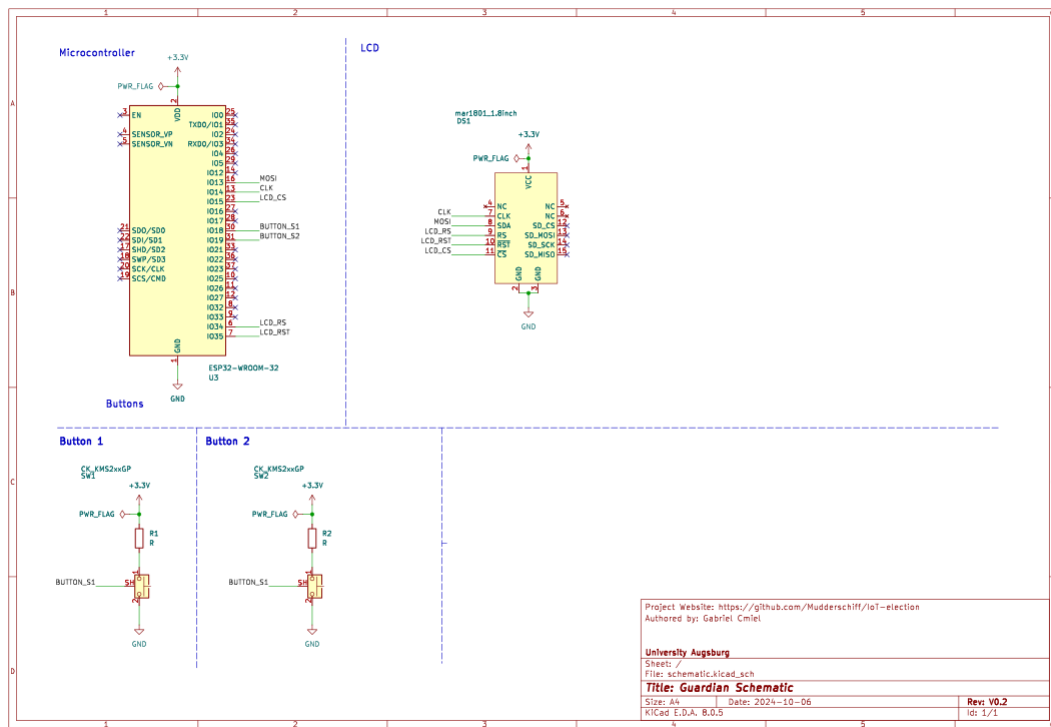
Figure 3.1: Schematic of the Guardian Prototype

on chip revision v3.0 or v3.1 which fixes some Hardware bugs **esp32-module-new**, **esp32-ds**, **esp32-errata**.

3.1.2 1,8" LCD Display Module

The 1,8" LCD Display Module has a resolution of 128x160 Pixels and is equipped with a ST7735R Display Driver. The module also contains a microSD Slot which won't be used in this project. **lcd**. The ESP-IDF component lvgl$_e$sp32$_d$rivers

3.1.3 Button Module

The Button Module is an integrated circuit that features 2 buttons and integrated pull-up resistors. **button-ds**

3.2 Implementation

3.2.1 Python Client

Our python client imports the python package ElectionGuard **python-reference**. The ElectionGuard Python package is a reference implementation of the ElectionGuard 1.0 specification. It covers the entire suite of functionality required to implement an end-to-end verifiable election as part of a voting system **eg-docs**.

In the proposed voting system the python client will act as the administrator, the ballot box and the encryption device of the election. Administrating the election requires loading and semantically verifiying the Election manifest before the election.

Before the election the python client load the Election Manifest for our election and semantically checks the data format required to conduct an ElectionGuard Election.

The python client loads the manifest file used for our election. In ElectionGuard an election manifest has to be semantically checked against the data format required to conduct an ElectionGuard Election.

The python client then generates the election keys and the election context. The election keys are used to encrypt the ballots and the election context is used to verify the election. The python client then encrypts the ballots and generates the proofs of the encryption. The encrypted ballots and the proofs are then sent to the ESP32. The ESP32 decrypts the ballots and verifies the proofs. The ESP32 then generates the proofs of the decryption and sends them back to the python client. The python client then verifies the proofs of the decryption and generates the proofs of the election. The proofs of the election are then sent to the ESP32. The ESP32 verifies the proofs of the election and sends the results back to the python client. The python client then verifies the results and outputs the final results of the election.

The tasks of the python client can be divided into three stages pre-election, intra-election and post-election.

The python client loads the manifest file used for our election. In ElectionGuard an election manifest has to be semantically checked against the data fromat required to conduct an ElectionGuard Election.

Espressif IoT Development Framework (ESP-IDF) is a software development framework intended for the development of Internet-of-Things (IoT) applications for the ESP32 board **esp-idf**. ESP-IDF consists of components written specifically for ESP-IDF as well as third-party libraries.**esp-idf** For example, the real-time operating system kernel used for ESP-IDF applications is the third-party component called FreeRTOS **esp-idf**. ESP-IDF projects use the same componment based approach and can be seen as an amalgamation of a number of components **esp-idf**.

ESP32 contains a hardware random number generator, values from it can be obtained using the APIs 'esp$_r$andom()$'and'esp_fill_random()'. If certain conditions are met, the RNG mix random.**esp-idf**

3.2.2 Model

ElectionGuard uses integer ElGamal within specific cryptographic equations three large integer operations are often performed all on very large integer values. These operations are modular exponentiation, modular multiplication, and modular addition. These operations can be performed by either importing specialiced tools to perform these large integer operations, or by implementing these operations from scratch. When implementing these operations from scratch, intermediate values can get excessively large often times it is necessary to perform special methods such as modular reduction to keep the values within a reasonable range. **eg-spec**. Another important operation within ElectionGuard is SHA-256 hash computations.
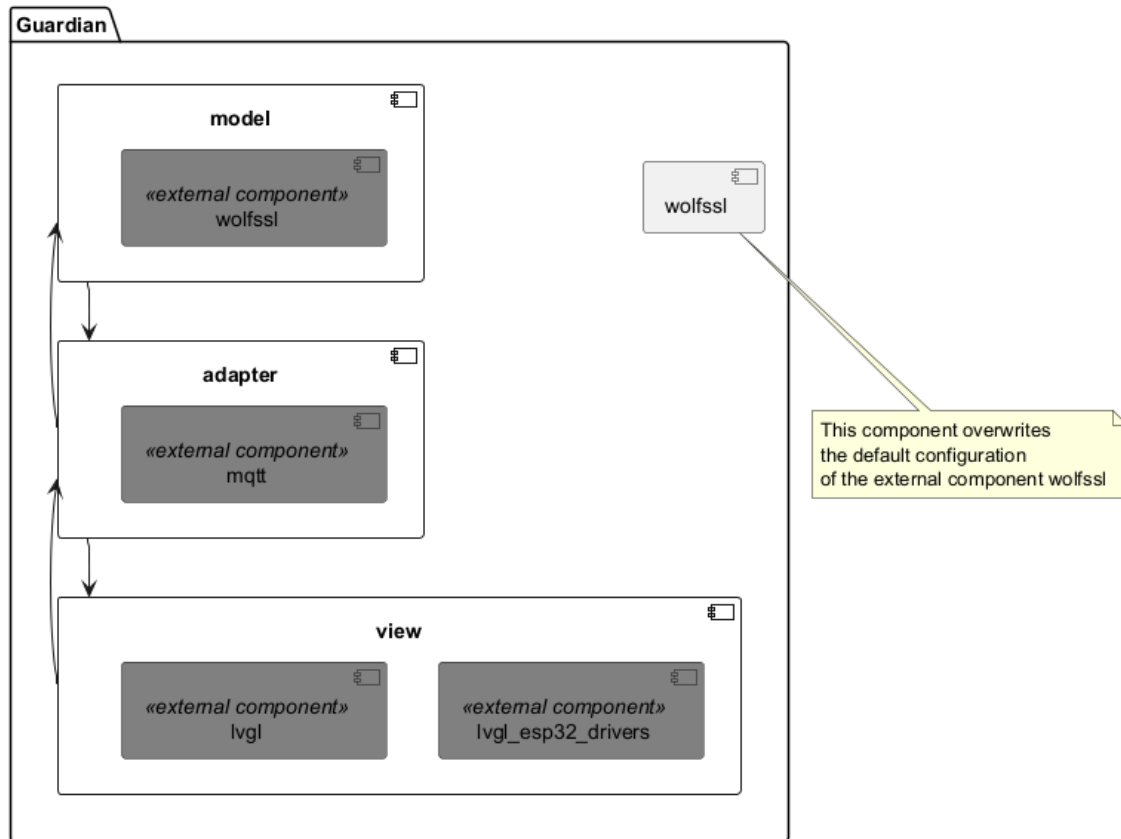
Figure 3.2:

Figure 3.3: Selected Python Classes from Reference Implementation for Guardian Functionality

Comparison with other Implementations

The modular exponentiation operation imposes the highest computational cost among all computations and is the limiting factor in any performance analysis. **eg-spec**. Using fast libraries for modular arithmetic is thus crucial to achieve good performance **eg-paper**. The Python reference implementation of ElectionGuard uses the C-code Gmpy2 library for large integer arithmetic. **eg-docs**. **esp32-ref** The C++ and Kotlin implementations of ElectionGuard uses HACL* C library for cryptographic primitives. **eg-docs**. Furthermore, the C++ implementation also uses pre-computed tables to speed up some modular exponentiations **eg-docs**. This is possible because most exponentiations have a fixed base, either the constant generator or the election public key. The pre-computed tables contain certain powers of these bases. **eg-docs**.

3.2.3 Feasability of the Python reference on ESP32

It could be possible to run the python reference on the ESP32 using Micropython. Micropython is an implementation of Python 3.x targetting microcontrollers and embedded systems. Functionalities of the python standard library are mirrored and "micro-ified" to fit with the limitations of microcontrollers such as memory and speed **micropython 9292199**. Apart from python the electionguard python reference requires the C-coded python library Gmpy2 for large integer arithmetic **eg-ref**. Disadvantages of this approach is that some needed modules, functions and classes may be missing from Micropython. **micropython**. Furthermore, MicroPython applications suffer from memory fragmentation and objects that grow in size **9292199**. Depending on task complexity and memory allocation MicroPython performs lower than the C implementation **9292199**. In a comparison of software based SHA-256 computation for the ESP32 the C implementation was faster than the MicroPython implementation by 45% **9292199**.

3.2.4 Feasability of the C++ reference on ESP32

ESP32 supports development of applications in C++. The ElectionGuard C++ implementation uses exception handling. C++ Exception handling has to be enabled in ESP-IDF. This will increase the application binary size by a few KB. Runtime Type Information (RTTI) can be left disables as dynamic cast conversion and the typeid operator are not used in the ElectionGuard C++ implementation. ESP32 also supports C++ threads however these are wrappers around C pthreads which in turn wrap FreeRTOS tasks. **espidf-docs**. After porting the C++ implementa-

tion into an ESP32 component we test the component using the included C Unit Tests specifically the test for elgamal encryption. The test crashes after calling $\text{pow}_mod_p()$. $\text{pow}_mod_p()$ is a function to speed up modular exponentiation by using a fixed-base lookup table.

Listing 3.1: FixedBaseTable Definition

```
typedef std::array<std::array<uint64_t[MAX_P_LEN], OrderBits>,
    TableLength> FixedBaseTable;
```

The FixedBaseTable is a 2d Array. $\text{MAX}_{PL}EN$ is the length of each $uint64_t$ array. $OrderBits$ is the number

The $\text{pow}_mod_p()$ implementation is tuned specifically for the following values: $borderbits = 256, kwindowsize = 8, mtablelength = 32$. Any changes to these values may impact the internal operation $bitwindowssize$ the order bits would be $256(2^8)$ **eg-spec**. The total size in bytes of the FixedBaseTable can bec

$$\text{FixedBaseTable Size} = \text{sizeof(uint64\_t)} \times \text{MAX\_P\_LEN} \times \text{OrderBits} \times \text{TableLength} \tag{3.1}$$

$\text{MAX}_{PL}EN$ is defined as 64. Substituting the above equation with the actual values we get $4MB$.

$$\text{FixedBaseTable Size} = 8 \times 64 \times 256 \times 32 = 4194304 \text{ bytes} = 4 \text{ MB} \tag{3.2}$$

If we calculate the size of the FixedBaseTable with the reduced baseline parameters

$$\text{FixedBaseTable Size} = 8 \times 48 \times 256 \times 32 = 3145728 \text{ bytes} = 3 \text{ MB} \tag{3.3}$$

The ESP32 has 320KB of DRAM to hold data. Due to a technical limitation the maximum memory available for dynamic allocation is 160KB. **esp32-ref**. This means that the FixedBaseTable is too large to be stored in the ESP32's memory. A decrease in window size k leads to smaller tables and thus less memory usage, but increases the number of multiplications. **eg-spec**. The ElectionGuard C++ implementation assumes Intel Atom class processor level performance and Raspberry Pi3 types of operating systems **eg-docs**. Base on these assumptions and the memory usage of the FixedBaseTable the ElectionGuard C++ implementation is not feasible on the ESP32.
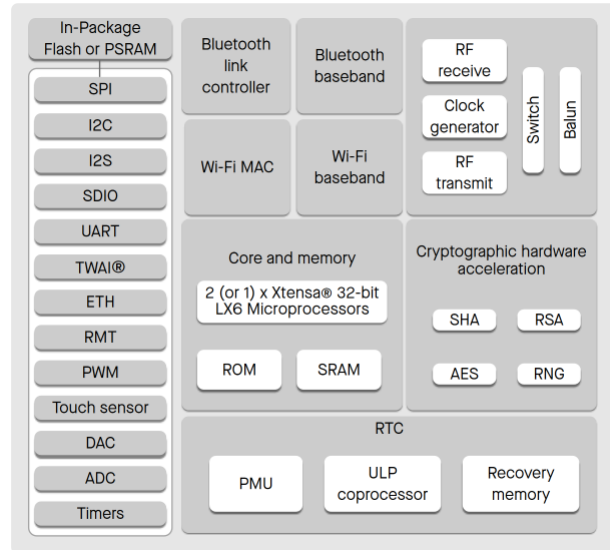
Figure 3.4: ESP32 Functional Block Diagram

3.2.5 C Implementation

From the previous sections we know that running both the Python and C++ reference implementations on the ESP32 is not feasible. In the case of Python there is the issue of possible memory fragmentation and performance and in the case of the C++ implementation the high memory requirements. Furthermore, the C++ implementation only contains the encryption library and thus only contains the intra-election steps as outlined in the background chapter. Interestingly, both implementations use a C coded library for modular arithmetic. For these reasons the author opts for a pure C implementation on the ESP32.

Performance of Cryptographic Operations

It is essential to test if the cryptographic algorithms used are viable in terms of execution time and memory consumption.

As seen in the Functional Block Diagram of the ESP32 **??** the ESP32 is equipped with Cryptographic hardware acceleration for general algorithms such as AES, SHA, RSA. The SHA Accelerator supports four algorithms: SHA-1, SHA-256, and SHA-384, and SHA-512.

The SHA Accelerator requires 60 to 100 clock cycles to process a message block and 8 to 20 clock cycles to calculate the final hash. For SHA256 a message block is

512 bits. The Accelerator only accepts one message block at a time furthermore the accelerator is unable to perform padding operations. Thus User software is expected to divide longer messages into 512-bit blocks and perform padding operations if needed. **esp32-ds**. The performance of SHA-256 with the SHA Accelerator and with a CPU frequency of 240MHz is 181.3% higher than without the SHA Accelerator. **performance-evaluation**. The performance is nearly 3 times faster with hardware acceleration compared to without. **performance-evaluation**.

Comparing the throughput performance of the SHA-256 with and without the SHA Accelerator the SHA Accelerator is has a 181.3

hardware accelerators of general algorithms, such as AES, SHA, RSA, and ECC. **esp32-ds**

libraries mbedtls: C library that implements cryptographic primitives Graphics Library: lvgl Communication: protobuf for serialization

3.2.6 Adapter

The under-specification

3.2.7 View

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

4 Conclusions

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

List of Figures

List of Tables

A Appendix

A.1 Subset of Reference Implementation for the Guardian functionality

A.2 Model Component

A.2.1 Model.c

Listing A.1: Business logic

```c
#include "model.h"
#include "esp_heap_caps.h"

/**
 * @brief Generates election key pair, proof, and polynomial
 * @param quorum: The number of guardians required to decrypt the
    election
 * @param key_pair: The election key pair
 * @return 0 on success, -1 on failure
 */
int generate_election_key_pair(int quorum, ElectionKeyPair *
   key_pair) {
    key_pair->public_key = NULL;
    NEW_MP_INT_SIZE(key_pair->public_key, 3072, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(key_pair->public_key, 3072);

    key_pair->private_key = NULL;
    NEW_MP_INT_SIZE(key_pair->private_key, 256, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(key_pair->private_key, 256);

    key_pair->polynomial.num_coefficients = quorum;
    key_pair->polynomial.coefficients = (Coefficient*)XMALLOC(
        quorum * sizeof(Coefficient), NULL, DYNAMIC_TYPE_BIGINT);
    if (key_pair->polynomial.coefficients == NULL) {
```

```c
        ESP_LOGE("Generate Election Key Pair", "Failed to allocate
            memory for coefficients");
        return -1;
    }
    generate_polynomial(&key_pair->polynomial);
    sp_copy(key_pair->polynomial.coefficients[0].value, key_pair->
        private_key);
    sp_copy(key_pair->polynomial.coefficients[0].commitment,
        key_pair->public_key);
    return 0;
}

/**
 * @brief Generate election partal key backup for sharing
 * @param sender_guardian_id: Owner of election key
 * @param sender_guardian_polynomial: The owner's Election
    polynomial
 * @param receiver_guardian_public_key: The receiving guardian's
    public key
 * @return PartialKeyBackup / Encrypted Coordinate
 */
int generate_election_partial_key_backup(ElectionKeyPair *sender,
    ElectionKeyPair *receiver, ElectionPartialKeyPairBackup *backup)
    {
    DECL_MP_INT_SIZE(coordinate, 256);
    NEW_MP_INT_SIZE(coordinate, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(coordinate, 256);

    DECL_MP_INT_SIZE(nonce, 256);
    NEW_MP_INT_SIZE(nonce, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(nonce, 256);

    DECL_MP_INT_SIZE(seed, 256);
    NEW_MP_INT_SIZE(seed, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(seed, 256);

    DECL_MP_INT_SIZE(id, 48);
    NEW_MP_INT_SIZE(id, 48, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(id, 48);
    sp_read_unsigned_bin(id, receiver->guardian_id, sizeof(receiver
        ->guardian_id));

    memcpy(backup->sender, sender->guardian_id, sizeof(sender->
        guardian_id));
    memcpy(backup->receiver, receiver->guardian_id, sizeof(receiver
        ->guardian_id));

    backup->encrypted_coordinate.pad = NULL;
    NEW_MP_INT_SIZE(backup->encrypted_coordinate.pad, 3072, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(backup->encrypted_coordinate.pad, 3072);
```

```c
    backup->encrypted_coordinate.data = NULL;
    NEW_MP_INT_SIZE(backup->encrypted_coordinate.data, 256, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(backup->encrypted_coordinate.data, 256);

    backup->encrypted_coordinate.mac = NULL;
    NEW_MP_INT_SIZE(backup->encrypted_coordinate.mac, 256, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(backup->encrypted_coordinate.mac, 256);

    compute_polynomial_coordinate(receiver->guardian_id, &sender->
        polynomial, coordinate);
    rand_q(nonce);
    hash(id, id, seed);
    hashed_elgamal_encrypt(coordinate, nonce, receiver->public_key,
        seed, &backup->encrypted_coordinate);
    sp_zero(nonce);
    sp_zero(seed);
    FREE_MP_INT_SIZE(coordinate, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(nonce, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(seed, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(id, NULL, DYNAMIC_TYPE_BIGINT);
    return 0;
}

/**
 * @brief Verify election partial key backup contain point on
    owners polynomial
 * @param guardian_id: Receiving guardian's identifier
 * @param sender_guardian_backup: Sender guardian's election
    partial key backup
 * @param sender_guardian_public_key: Sender guardian's election
    public key
 * @param receiver_guardian_keys: Receiving guardian's key pair
 */
int verify_election_partial_key_backup(ElectionKeyPair *receiver,
    ElectionKeyPair *sender, ElectionPartialKeyPairBackup *backup,
    ElectionPartialKeyVerification *verification) {
    DECL_MP_INT_SIZE(encryption_seed, 256);
    NEW_MP_INT_SIZE(encryption_seed, 256, NULL, DYNAMIC_TYPE_BIGINT
        );
    INIT_MP_INT_SIZE(encryption_seed, 256);

    DECL_MP_INT_SIZE(coordinate, 3072);
    NEW_MP_INT_SIZE(coordinate, 3072, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(coordinate, 3072);

    DECL_MP_INT_SIZE(gid, 48);
    NEW_MP_INT_SIZE(gid, 48, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(gid, 48);
    DECL_MP_INT_SIZE(bid, 48);
    NEW_MP_INT_SIZE(bid, 48, NULL, DYNAMIC_TYPE_BIGINT);
```

```
    INIT_MP_INT_SIZE(bid, 48);
    sp_read_unsigned_bin(gid, receiver->guardian_id, sizeof(
        receiver->guardian_id));
    sp_read_unsigned_bin(bid, backup->receiver, sizeof(backup->
        receiver));

    memcpy(verification->sender, backup->sender, sizeof(backup->
        sender));
    memcpy(verification->receiver, backup->receiver, sizeof(backup
        ->receiver));
    memcpy(verification->verifier, receiver->guardian_id, sizeof(
        receiver->guardian_id));
    verification->verified = false;
    //get_backup_seed()
    hash(gid, bid, encryption_seed);
    // decrypt encrypted_coordinate
    hashed_elgamal_decrypt(&backup->encrypted_coordinate, receiver
        ->private_key, encryption_seed, coordinate);
    verification->verified = verify_polynomial_coordinate(backup->
        receiver, &sender->polynomial, coordinate);

    sp_zero(encryption_seed);
    FREE_MP_INT_SIZE(encryption_seed, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(coordinate, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(gid, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(bid, NULL, DYNAMIC_TYPE_BIGINT);
    return 0;
}

int combine_election_public_keys(ElectionKeyPair *guardian,
    ElectionKeyPair *pubkey_map, size_t count, ElectionJointKey *
    joint_key) {
    joint_key->joint_key = NULL;
    NEW_MP_INT_SIZE(joint_key->joint_key, 3072, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(joint_key->joint_key, 3072);

    joint_key->commitment_hash = NULL;
    NEW_MP_INT_SIZE(joint_key->commitment_hash, 256, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(joint_key->commitment_hash, 256);

    ElectionKeyPair *extend = (ElectionKeyPair*)XMALLOC((count + 1)
        * sizeof(ElectionKeyPair), NULL, DYNAMIC_TYPE_BIGINT);
    memcpy(extend, pubkey_map, count * sizeof(ElectionKeyPair));
    extend[count] = *guardian;

    elgamal_combine_public_keys(extend, count + 1, joint_key);
    hash_keys(extend, count + 1, joint_key);
    return 0;
}
```

```c
int compute_decryption_share(ElectionKeyPair *guardian,
    CiphertextTally *ciphertally, DecryptionShare *share) {
    share->object_id = NULL;
    share->object_id = strdup(ciphertally->object_id);
    share->public_key = NULL;

    memcpy(share->guardian_id, guardian->guardian_id, sizeof(
        guardian->guardian_id));

    NEW_MP_INT_SIZE(share->public_key, 3072, NULL,
        DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(share->public_key, 3072);
    sp_copy(guardian->public_key, share->public_key);

    share->num_contest = ciphertally->num_contest;
    share->contests = (CiphertextDecryptionContest*)XMALLOC(
        ciphertally->num_contest * sizeof(
        CiphertextDecryptionContest), NULL, DYNAMIC_TYPE_BIGINT);
    for (int i = 0; i < ciphertally->num_contest; i++) {
        compute_decryption_share_for_contest(guardian, &ciphertally
            ->contests[i], ciphertally->base_hash , &share->contests
            [i]);
    }

    return 0;
}
```

A.3 View Component

```c
#include "view.h"

/*********************
 *      DEFINES
 *********************/
#define TAG "GUI"
#define LV_TICK_PERIOD_MS 1

void guiTask(void *pvParameter);


/* Creates a semaphore to handle concurrent call to lvgl stuff
 * If you wish to call *any* lvgl function from other threads/tasks
 * you should lock on the very same semaphore! */
SemaphoreHandle_t xGuiSemaphore;

void guiTask(void *pvParameter) {

    xGuiSemaphore = xSemaphoreCreateMutex();

    lv_init();
```

```c
    /* Initialize SPI or I2C bus used by the drivers */
    lvgl_driver_init();
    ESP_LOGI(TAG, "OK so far!");

    while(1) {
        vTaskDelay(100 / portTICK_RATE_MS);
    }
}
```

A.4 Adapter Component

```c
#include "adapter.h"

static const char *TAG = "Adapter";
uint8_t mac[6] = {0};

int pubkey_count = 0;
int backup_count = 0;
//int verification_count = 0;
int max_guardians;
int quorum;

ElectionKeyPair guardian;

ElectionKeyPair *pubkey_map;
ElectionPartialKeyPairBackup *backup_map;

void publish_public_key(esp_mqtt_client_handle_t client, const char
    *data, int data_len);
void handle_pubkeys(esp_mqtt_client_handle_t client, const char *
   data, int data_len);
void handle_backups(esp_mqtt_client_handle_t client, const char *
   data, int data_len);
void handle_challenge(esp_mqtt_client_handle_t client, const char *
   data, int data_len);
void handle_ciphertext_tally(esp_mqtt_client_handle_t client, const
    char *data, int data_len);

void add_key_pair(ElectionKeyPair *key_pair);
ElectionKeyPair* find_key_pair(uint8_t *guardian_id);
void delete_key_pair(uint8_t *guardian_id);

void add_backup(ElectionPartialKeyPairBackup *backup);
ElectionPartialKeyPairBackup* find_backup(uint8_t *guardian_id);
void delete_backup(uint8_t *guardian_id);


void log_error_if_nonzero(const char *message, int error_code)
{
    if (error_code != 0)
    {
```

```c
            ESP_LOGE(TAG, "Last error %s: 0x%x", message, error_code);
    }
}

void log_heap_info() {
    ESP_LOGI(TAG, "Heap summary:");
    ESP_LOGI(TAG, "  Free heap size: %d", heap_caps_get_free_size(
        MALLOC_CAP_8BIT));
    ESP_LOGI(TAG, "  Minimum free heap size: %d",
        heap_caps_get_minimum_free_size(MALLOC_CAP_8BIT));
    ESP_LOGI(TAG, "  Largest free block: %d",
        heap_caps_get_largest_free_block(MALLOC_CAP_8BIT));
}


/*
 * @brief Event handler registered to receive MQTT events
 *
 *  This function is called by the MQTT client event loop.
 *
 * @param handler_args user data registered to the event.
 * @param base Event base for the handler(always MQTT Base in this
     example).
 * @param event_id The id for the received event.
 * @param event_data The data for the event,
     esp_mqtt_event_handle_t.
 */
void mqtt_event_handler(void *handler_args, esp_event_base_t base,
    int32_t event_id, void *event_data)
{
    ESP_LOGD(TAG, "Event dispatched from event loop base=%s,
        event_id=%" PRIi32 "", base, event_id);
    esp_mqtt_event_handle_t event = event_data;
    esp_mqtt_client_handle_t client = event->client;
    //int msg_id;
    switch ((esp_mqtt_event_id_t)event_id)
    {
    case MQTT_EVENT_BEFORE_CONNECT:
        ESP_LOGI(TAG, "MQTT_EVENT_BEFORE_CONNECT");
        // Get the MAC address of the device and set it as the
            guardian_id
        esp_efuse_mac_get_default(mac);
        memcpy(guardian.guardian_id, mac, 6);
        break;
    case MQTT_EVENT_CONNECTED:
        esp_mqtt_client_subscribe(client, "ceremony_details", 1);
        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
        break;
    case MQTT_EVENT_SUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->
```

```c
                msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->
            msg_id);
        if (strncmp(event->topic, "pub_keys", event->topic_len) ==
            0) {
            // Generate all backups and publish them
            ESP_LOGI(TAG, "Unsubscribed from pub_keys topic");
            for (int i = 0; i < max_guardians; i++) {
                ElectionKeyPair *sender = &pubkey_map[i];
                ElectionPartialKeyPairBackup backup;
                void *buffer;
                size_t len;

                generate_election_partial_key_backup(sender, &
                    guardian, &backup);
                buffer = serialize_election_partial_key_backup(&
                    backup, &len);
                esp_mqtt_client_publish(client, "backups", buffer,
                    len, 2, 0);

                free(buffer);
                free_ElectionPartialKeyPairBackup(&backup);
            }
        } else if(strncmp(event->topic, "backups", event->topic_len
            ) == 0) {
            // Verify all backups. If verify fails send challenge
            bool all_verified = true;
            for (int i = 0; i < max_guardians; i++) {
                ElectionPartialKeyPairBackup *backup = &backup_map[
                    i];
                ElectionPartialKeyVerification verification;
                void *buffer;
                size_t len;

                verify_election_partial_key_backup(&guardian,
                    find_key_pair(backup->sender), backup, &
                    verification);
                if(verification.verified == 0)
                {
                    ESP_LOGI(TAG, "Proof failed");
                    // send challenge
                    buffer =
                        serialize_election_partial_key_verification
                        (&verification, &len);
                    esp_mqtt_client_publish(client, "challenge",
                        buffer, len, 2, 0);
                    free(buffer);
                    all_verified = false;
                } else {
                    ESP_LOGI(TAG, "Proof verified");
```

```c
                }
            }
            if(all_verified)
            {
                ESP_LOGI(TAG, "All backups verified");
                ElectionJointKey joint_key;
                combine_election_public_keys(&guardian, pubkey_map,
                    pubkey_count, &joint_key);
                int size = sp_unsigned_bin_size(joint_key.joint_key
                    ) + sp_unsigned_bin_size(joint_key.
                    commitment_hash);
                byte *buffer = malloc(size);
                sp_to_unsigned_bin(joint_key.joint_key, buffer);
                sp_to_unsigned_bin_at_pos(sp_unsigned_bin_size(
                    joint_key.joint_key), joint_key.commitment_hash,
                     buffer);
                esp_mqtt_client_publish(client, "joint_key", (char
                    *)buffer, size, 2, 0);
                esp_mqtt_client_subscribe(client, "ciphertally", 1)
                    ;
                free(buffer);
            }

    }
    break;
case MQTT_EVENT_PUBLISHED:
    ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->
        msg_id);
    break;
case MQTT_EVENT_DATA:
    ESP_LOGI(TAG, "MQTT_EVENT_DATA");
    char topic[20];
    snprintf(topic, event->topic_len + 1, "%.*s", event->
        topic_len, event->topic);
    ESP_LOGI(TAG, "Topic: %s", topic);
    if(strncmp(topic, "ceremony_details", event->topic_len) ==
        0)
    {
        if(sscanf(event->data, "%d,%d", &quorum, &max_guardians
            ) == 2) {
            ESP_LOGI(TAG, "Received Ceremony Details");
            ESP_LOGI(TAG, "Quorum: %d, Max Guardians: %d",
                quorum, max_guardians);
            // Exclude self from guardian count
            max_guardians--;
            pubkey_map = (ElectionKeyPair*)malloc(max_guardians
                 * sizeof(ElectionKeyPair));
            backup_map = (ElectionPartialKeyPairBackup*)malloc(
                max_guardians * sizeof(
                ElectionPartialKeyPairBackup));
            esp_mqtt_client_unsubscribe(client, "
                ceremony_details");
```

```
                esp_mqtt_client_subscribe(client, "pub_keys", 1);
                esp_mqtt_client_subscribe(client, "backups", 1);
                esp_mqtt_client_subscribe(client, "challenge", 1);
                publish_public_key(client, event->data, event->
                    data_len);
            }
        }
        else if(strncmp(topic, "pub_keys", event->topic_len) == 0)
        {
            ESP_LOGI(TAG, "Received Public Key");
            handle_pubkeys(client, event->data, event->data_len);
        }
        else if(strncmp(topic, "backups", event->topic_len) == 0)
        {
            ESP_LOGI(TAG, "Received Backup");
            handle_backups(client, event->data, event->data_len);
        }
        else if(strncmp(topic, "challenge", event->topic_len) == 0)
        {
            ESP_LOGI(TAG, "Received Challenge");
            handle_challenge(client, event->data, event->data_len);
        } else if (strncmp(topic, "ciphertally", event->topic_len)
            == 0)
        {
            ESP_LOGI(TAG, "Received ciphertext tally");
            handle_ciphertext_tally(client, event->data, event->
                data_len);
        }
        else {
            ESP_LOGI(TAG, "Unknown topic");
        }
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
        if (event->error_handle->error_type ==
            MQTT_ERROR_TYPE_TCP_TRANSPORT)
        {
            log_error_if_nonzero("reported from esp-tls", event->
                error_handle->esp_tls_last_esp_err);
            log_error_if_nonzero("reported from tls stack", event->
                error_handle->esp_tls_stack_err);
            log_error_if_nonzero("captured as transport's socket
                errno", event->error_handle->
                esp_transport_sock_errno);
            ESP_LOGI(TAG, "Last errno string (%s)", strerror(event
                ->error_handle->esp_transport_sock_errno));
        }
        break;
    default:
        ESP_LOGI(TAG, "Other event id:%d", event->event_id);
        break;
    }
```

```c
}

void publish_public_key(esp_mqtt_client_handle_t client, const char
    *data, int data_len)
{
    void *buffer;
    size_t len;
    generate_election_key_pair(quorum, &guardian);
    buffer = serialize_election_key_pair(&guardian, &len);
    esp_mqtt_client_publish(client, "pub_keys", buffer, len, 2, 0);
    ESP_LOGI(TAG, "Sent Public Key");
    free(buffer);
}

void handle_pubkeys(esp_mqtt_client_handle_t client, const char *
    data, int data_len)
{
    ElectionKeyPair sender;
    deserialize_election_key_pair((uint8_t*)data, data_len, &sender
        );

    if(memcmp(sender.guardian_id, mac, 6) == 0)
    {
        ESP_LOGI(TAG, "Received own public key");
        free_ElectionKeyPair(&sender);
        return;
    }

    if(find_key_pair(sender.guardian_id) == NULL)
    {
        ESP_LOGI(TAG, "Adding Public Key");
        add_key_pair(&sender);
        if(pubkey_count == max_guardians)
        {
            ESP_LOGI(TAG, "All Public Keys received");
            esp_mqtt_client_unsubscribe(client, "pub_keys");
        }
    } else {
        ESP_LOGI(TAG, "Public Key already exists");
        free_ElectionKeyPair(&sender);
    }
}

void handle_backups(esp_mqtt_client_handle_t client, const char *
    data, int data_len)
{
    ElectionPartialKeyPairBackup backup;
    deserialize_election_partial_key_backup((uint8_t*)data,
        data_len, &backup);

    if(memcmp(backup.receiver, mac, 6) != 0)
    {
```

```
        ESP_LOGI(TAG, "Backup not intended for this guardian");
        free_ElectionPartialKeyPairBackup(&backup);
        return;
    }

    if(find_backup(backup.sender) == NULL)
    {
        ESP_LOGI(TAG, "Adding Backup");
        add_backup(&backup);
        if(backup_count == max_guardians)
        {
            ESP_LOGI(TAG, "All Backups received");
            esp_mqtt_client_unsubscribe(client, "backups");
        }
    } else {
        ESP_LOGI(TAG, "Backup already exists");
        free_ElectionPartialKeyPairBackup(&backup);
    }
}

void handle_challenge(esp_mqtt_client_handle_t client, const char *
    data, int data_len)
{
    return;
}

void handle_ciphertext_tally(esp_mqtt_client_handle_t client, const
    char *data, int data_len)
{
    CiphertextTally tally;
    DecryptionShare share;
    deserialize_ciphertext_tally((uint8_t*)data, data_len, &tally);
    compute_decryption_share(&guardian, &tally, &share);
    void *buffer;
    size_t len;
    buffer = serialize_DecryptionShare(&share, &len);
    esp_mqtt_client_publish(client, "decryption_share", buffer, len
        , 2, 0);
}

// Function to add an entry to the key pair map
void add_key_pair(ElectionKeyPair *key_pair) {
    if (pubkey_count < max_guardians) {
        pubkey_map[pubkey_count++] = *key_pair;
    } else {
        printf("Key pair map is full\n");
    }
}

// Function to find an entry in the key pair map
ElectionKeyPair* find_key_pair(uint8_t *guardian_id) {
    for (int i = 0; i < max_guardians; i++) {
```

```c
        if (memcmp(pubkey_map[i].guardian_id, guardian_id, 6) == 0)
            {
            return &pubkey_map[i];
        }
    }
    return NULL;
}

// Function to delete an entry from the key pair map
void delete_key_pair(uint8_t *guardian_id) {
    ElectionKeyPair *key_pair = find_key_pair(guardian_id);
    if (key_pair != NULL) {
        int index = key_pair - pubkey_map;
        free_ElectionKeyPair(&pubkey_map[index]);
        // Shift remaining elements to the left
        memmove(&pubkey_map[index], &pubkey_map[index + 1], (
            pubkey_count - index - 1) * sizeof(ElectionKeyPair));
        pubkey_count--;
    } else {
        printf("Guardian ID not found\n");
    }
}

// Function to add an entry to the key pair map
void add_backup(ElectionPartialKeyPairBackup *backup) {
    if (backup_count < max_guardians) {
        backup_map[backup_count++] = *backup;
    } else {
        printf("Key pair map is full\n");
    }
}

ElectionPartialKeyPairBackup* find_backup(uint8_t *guardian_id) {
    for (int i = 0; i < max_guardians; i++) {
        if (memcmp(backup_map[i].sender, guardian_id, 6) == 0) {
            return &backup_map[i];
        }
    }
    return NULL;
}

void delete_backup(uint8_t *guardian_id) {
    ElectionPartialKeyPairBackup *backup = find_backup(guardian_id)
        ;
    if (backup != NULL) {
        int index = backup - backup_map;
        free_ElectionPartialKeyPairBackup(&backup_map[index]);
        // Shift remaining elements to the left
        memmove(&backup_map[index], &backup_map[index + 1], (
            backup_count - index - 1) * sizeof(
            ElectionPartialKeyPairBackup));
        backup_count--;
```

```c
        } else {
            printf("Guardian ID not found\n");
        }
}

void mqtt_app_start(void)
{
    esp_mqtt_client_config_t mqtt_cfg = {
        .broker.address.uri = "mqtt://192.168.12.1:1883",
        .session.last_will = {
            .topic = "guardian_status",
            .qos = 2,
            .retain = 1,
            .msg = "Guardian has disconnected",
        },
        .buffer.size = 4096,
    };
    esp_mqtt_client_handle_t client = esp_mqtt_client_init(&
        mqtt_cfg);
    /* The last argument may be used to pass data to the event
        handler, in this example mqtt_event_handler */
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID,
        mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);

}
```