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:	20th 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.

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

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**.

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.

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.

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 veWrsions with higher integrated SPI flash sizes (8MB or 16MB) **esp32-module-new**, **esp32-module**.

3.1.2 Cryptographic Hardware accelerators

3.2 Implementation/Software Architecture

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

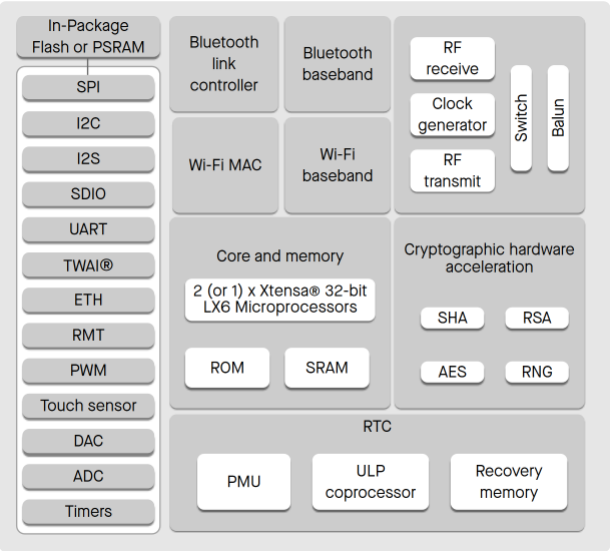

Figure 3.1: ESP32 Functional Block Diagram

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 $\mathrm{esp}_random()and esp_fill_random().If certainconditionsaremet, theRNGmixes random.$**esp-idf**

Application Protocols ESP-MQTT an implementation of MQTT protocol client.

esp-idf

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 Micropy-

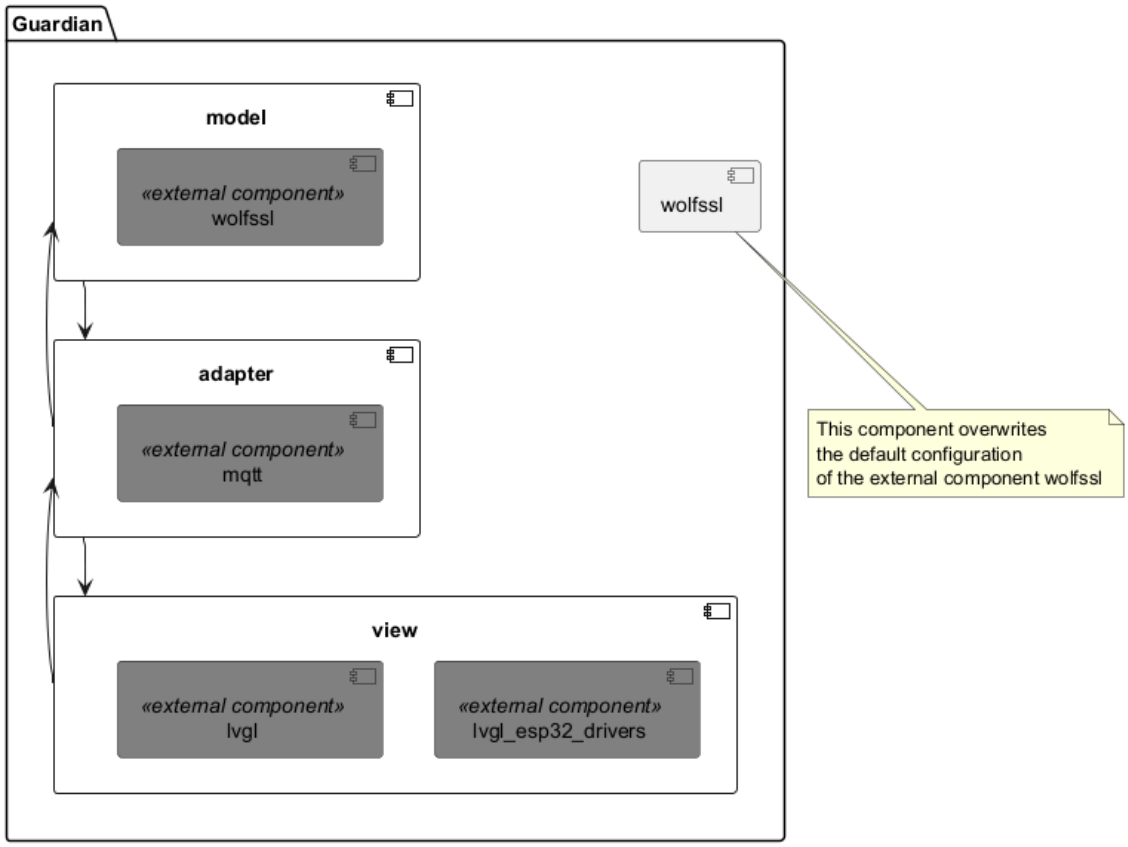

Figure 3.2:

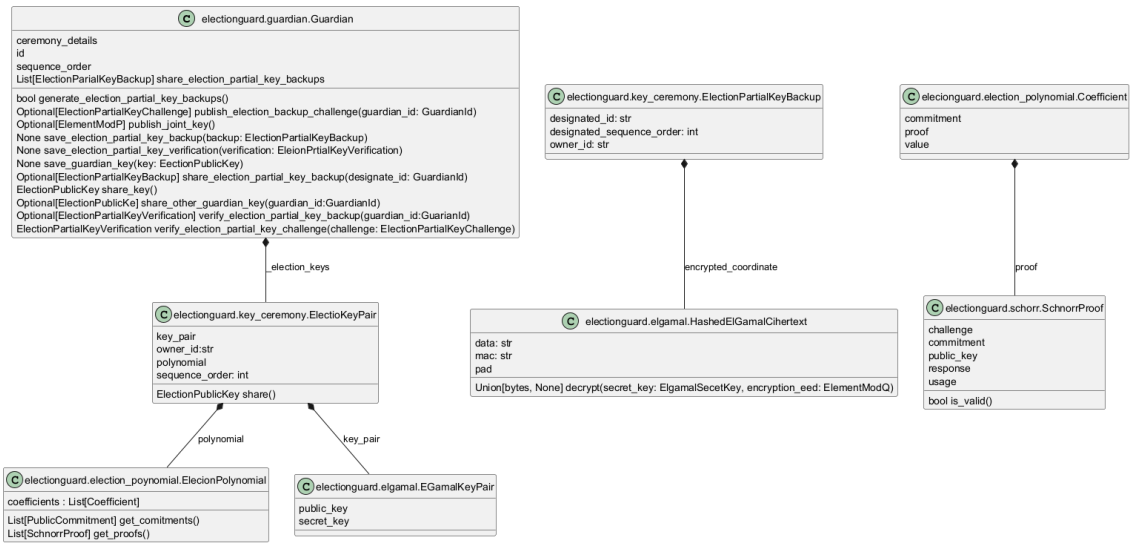

Figure 3.3: Selected Python Classes from Reference Implementation for Guardian Functionality

thon. [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

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 3.1: Beispieltabelle

Hier wird die Beispieltabelle **??** referenziert.

Figure 3.4: Bildunterschrift mit Quellenangabe **lcd**

Hier wird die Beispielbild **??** referenziert.

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

Hier wird die Beispielformel **??** referenziert.

kursiv, **fett**, unterstrichen

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

```
/* wolfSSL */
/* Always include wolfcrypt/settings.h before any other wolfSSL
   file.    */
/* Reminder: settings.h pulls in user_settings.h; don't include it
   here. */
#ifdef WOLFSSL_USER_SETTINGS
    #include <wolfssl/wolfcrypt/settings.h>
    #ifndef WOLFSSL_ESPIDF
        #warning "Problem with wolfSSL user_settings."
        #warning "Check components/wolfssl/include"
    #endif
    #include <wolfssl/version.h>
    #include <wolfssl/wolfcrypt/types.h>
    #include <wolfcrypt/test/test.h>
    #include <wolfssl/wolfcrypt/port/Espressif/esp-sdk-lib.h>
    #include <wolfssl/wolfcrypt/port/Espressif/esp32-crypt.h>
#else
    /* Define WOLFSSL_USER_SETTINGS project wide for settings.h to
       include   */
    /* wolfSSL user settings in ./components/wolfssl/include/
       user_settings.h */
    #error "Missing WOLFSSL_USER_SETTINGS in CMakeLists or Makefile
       :\
    CFLAGS +=-DWOLFSSL_USER_SETTINGS"
#endif

#include "model.h"
```

```
static const unsigned char p_3072[] = {
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0x93, 0xC4, 0x67, 0xE3, 0x7D, 0xB0, 0xC7, 0xA4,
    0xD1, 0xBE, 0x3F, 0x81, 0x01, 0x52, 0xCB, 0x56,
    0xA1, 0xCE, 0xCC, 0x3A, 0xF6, 0x5C, 0xC0, 0x19,
    0x0C, 0x03, 0xDF, 0x34, 0x70, 0x9A, 0xFF, 0xBD,
    0x8E, 0x4B, 0x59, 0xFA, 0x03, 0xA9, 0xF0, 0xEE,
    0xD0, 0x64, 0x9C, 0xCB, 0x62, 0x10, 0x57, 0xD1,
    0x10, 0x56, 0xAE, 0x91, 0x32, 0x13, 0x5A, 0x08,
    0xE4, 0x3B, 0x46, 0x73, 0xD7, 0x4B, 0xAF, 0xEA,
    0x58, 0xDE, 0xB8, 0x78, 0xCC, 0x86, 0xD7, 0x33,
    0xDB, 0xE7, 0xBF, 0x38, 0x15, 0x4B, 0x36, 0xCF,
    0x8A, 0x96, 0xD1, 0x56, 0x78, 0x99, 0xAA, 0xAE,
    0x0C, 0x09, 0xD4, 0xC8, 0xB6, 0xB7, 0xB8, 0x6F,
    0xD2, 0xA1, 0xEA, 0x1D, 0xE6, 0x2F, 0xF8, 0x64,
    0x3E, 0xC7, 0xC2, 0x71, 0x82, 0x79, 0x77, 0x22,
    0x5E, 0x6A, 0xC2, 0xF0, 0xBD, 0x61, 0xC7, 0x46,
    0x96, 0x15, 0x42, 0xA3, 0xCE, 0x3B, 0xEA, 0x5D,
    0xB5, 0x4F, 0xE7, 0x0E, 0x63, 0xE6, 0xD0, 0x9F,
    0x8F, 0xC2, 0x86, 0x58, 0xE8, 0x05, 0x67, 0xA4,
    0x7C, 0xFD, 0xE6, 0x0E, 0xE7, 0x41, 0xE5, 0xD8,
    0x5A, 0x7B, 0xD4, 0x69, 0x31, 0xCE, 0xD8, 0x22,
    0x03, 0x65, 0x59, 0x49, 0x64, 0xB8, 0x39, 0x89,
    0x6F, 0xCA, 0xAB, 0xCC, 0xC9, 0xB3, 0x19, 0x59,
    0xC0, 0x83, 0xF2, 0x2A, 0xD3, 0xEE, 0x59, 0x1C,
    0x32, 0xFA, 0xB2, 0xC7, 0x44, 0x8F, 0x2A, 0x05,
    0x7D, 0xB2, 0xDB, 0x49, 0xEE, 0x52, 0xE0, 0x18,
    0x27, 0x41, 0xE5, 0x38, 0x65, 0xF0, 0x04, 0xCC,
    0x8E, 0x70, 0x4B, 0x7C, 0x5C, 0x40, 0xBF, 0x30,
    0x4C, 0x4D, 0x8C, 0x4F, 0x13, 0xED, 0xF6, 0x04,
    0x7C, 0x55, 0x53, 0x02, 0xD2, 0x23, 0x8D, 0x8C,
    0xE1, 0x1D, 0xF2, 0x42, 0x4F, 0x1B, 0x66, 0xC2,
    0xC5, 0xD2, 0x38, 0xD0, 0x74, 0x4D, 0xB6, 0x79,
    0xAF, 0x28, 0x90, 0x48, 0x70, 0x31, 0xF9, 0xC0,
    0xAE, 0xA1, 0xC4, 0xBB, 0x6F, 0xE9, 0x55, 0x4E,
    0xE5, 0x28, 0xFD, 0xF1, 0xB0, 0x5E, 0x5B, 0x25,
    0x62, 0x23, 0xB2, 0xF0, 0x92, 0x15, 0xF3, 0x71,
    0x9F, 0x9C, 0x7C, 0xCC, 0x69, 0xDE, 0xD4, 0xE5,
    0x30, 0xA6, 0xEC, 0x94, 0x0C, 0x45, 0x31, 0x4D,
    0x16, 0xD3, 0xD8, 0x64, 0xB4, 0xA8, 0x93, 0x4F,
    0x8B, 0x87, 0xC5, 0x2A, 0xFA, 0x09, 0x61, 0xA0,
    0xA6, 0xC5, 0xEE, 0x4A, 0x35, 0x37, 0x77, 0x73,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
};
```

```
static const unsigned char r_3072[] __attribute__((unused)) = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xBC,
    0x93, 0xC4, 0x67, 0xE3, 0x7D, 0xB0, 0xC7, 0xA4,
    0xD1, 0xBE, 0x3F, 0x81, 0x01, 0x52, 0xCB, 0x56,
    0xA1, 0xCE, 0xCC, 0x3A, 0xF6, 0x5C, 0xC0, 0x19,
    0x0C, 0x03, 0xDF, 0x34, 0x70, 0x9B, 0x8A, 0xF6,
    0xA6, 0x4C, 0x0C, 0xED, 0xCF, 0x2D, 0x55, 0x9D,
    0xA9, 0xD9, 0x7F, 0x09, 0x5C, 0x30, 0x76, 0xC6,
    0x86, 0x03, 0x76, 0x19, 0x14, 0x8D, 0x2C, 0x86,
    0xC3, 0x17, 0x10, 0x2A, 0xFA, 0x21, 0x48, 0x03,
    0x1F, 0x04, 0x44, 0x0A, 0xC0, 0xFF, 0x0C, 0x9A,
    0x41, 0x7A, 0x89, 0x21, 0x25, 0x12, 0xE7, 0x60,
    0x7B, 0x25, 0x01, 0xDA, 0xA4, 0xD3, 0x8A, 0x2C,
    0x14, 0x10, 0xC4, 0x83, 0x61, 0x49, 0xE2, 0xBD,
    0xB8, 0xC8, 0x26, 0x0E, 0x62, 0x7C, 0x46, 0x46,
    0x96, 0x3E, 0xFF, 0xE9, 0xE1, 0x6E, 0x49, 0x5D,
    0x48, 0xBD, 0x21, 0x5C, 0x6D, 0x8E, 0xC9, 0xD1,
    0x66, 0x76, 0x57, 0xA2, 0xA1, 0xC8, 0x50, 0x6F,
    0x21, 0x13, 0xFF, 0xAD, 0x19, 0xA6, 0xB2, 0xBC,
    0x7C, 0x45, 0x76, 0x04, 0x56, 0x71, 0x91, 0x83,
    0x30, 0x9F, 0x87, 0x4B, 0xC9, 0xAC, 0xE5, 0x70,
    0xFF, 0xDA, 0x87, 0x7A, 0xA2, 0xB2, 0x3A, 0x2D,
    0x6F, 0x29, 0x1C, 0x15, 0x54, 0xCA, 0x2E, 0xB1,
    0x2F, 0x12, 0xCD, 0x00, 0x9B, 0x8B, 0x87, 0x34,
    0xA6, 0x4A, 0xD5, 0x1E, 0xB8, 0x93, 0xBD, 0x89,
    0x17, 0x50, 0xB8, 0x51, 0x62, 0x24, 0x1D, 0x90,
    0x8F, 0x0C, 0x97, 0x09, 0x87, 0x97, 0x58, 0xE7,
    0xE8, 0x23, 0x3E, 0xAB, 0x3B, 0xF2, 0xD6, 0xAB,
    0x53, 0xAF, 0xA3, 0x2A, 0xA1, 0x53, 0xAD, 0x66,
    0x82, 0xE5, 0xA0, 0x64, 0x88, 0x97, 0xC9, 0xBE,
    0x18, 0xA0, 0xD5, 0x0B, 0xEC, 0xE0, 0x30, 0xC3,
    0x43, 0x23, 0x36, 0xAD, 0x91, 0x63, 0xE3, 0x3F,
    0x8E, 0x7D, 0xAF, 0x49, 0x8F, 0x14, 0xBB, 0x28,
    0x52, 0xAF, 0xFA, 0x81, 0x48, 0x41, 0xEB, 0x18,
    0xDD, 0x5F, 0x0E, 0x89, 0x51, 0x6D, 0x55, 0x77,
    0x76, 0x28, 0x5C, 0x16, 0x07, 0x1D, 0x21, 0x11,
    0x94, 0xEE, 0x1C, 0x3F, 0x34, 0x64, 0x20, 0x36,
    0xAB, 0x88, 0x6E, 0x3E, 0xC2, 0x89, 0x66, 0x40,
    0x9F, 0xD4, 0xA7, 0xF5, 0x29, 0xFD, 0x4A, 0x7F,
    0x52, 0x9F, 0xD4, 0xA7, 0xF5, 0x29, 0xFD, 0x4A,
    0x7F, 0x52, 0x9F, 0xD4, 0xA7, 0xF5, 0x29, 0xFD,
    0x4A, 0x7F, 0x52, 0x9F, 0xD4, 0xA7, 0xF5, 0x2A
};


static const unsigned char g_3072[] = {
    0xAF, 0x8D, 0xC2, 0x05, 0x79, 0x63, 0xC6, 0xC3,
    0x64, 0x11, 0x9C, 0x01, 0x4A, 0x27, 0x68, 0x6B,
```

```
    0xA7, 0x80, 0x57, 0x67, 0x48, 0xB7, 0x2F, 0x67,
    0x0C, 0x4A, 0x5D, 0x4C, 0x3F, 0xAC, 0x1E, 0x22,
    0x8B, 0x84, 0xFB, 0xA8, 0x8C, 0x4E, 0xAF, 0x94,
    0xDF, 0x98, 0x75, 0x5C, 0x6C, 0x73, 0x61, 0x1B,
    0xB5, 0x4A, 0x14, 0xA6, 0xE2, 0x32, 0xD2, 0x38,
    0xC9, 0x17, 0xDA, 0x76, 0xD8, 0xA6, 0x2B, 0x70,
    0x83, 0x7A, 0x15, 0xEE, 0xC1, 0x11, 0x0C, 0x11,
    0x25, 0x61, 0xAB, 0x0E, 0xAE, 0x9E, 0x11, 0xDD,
    0xCE, 0xC6, 0x1F, 0x2B, 0xBD, 0x54, 0xBB, 0x76,
    0x2F, 0xC9, 0x03, 0x49, 0x4E, 0xF2, 0x1F, 0x0F,
    0x33, 0x8F, 0xE2, 0x65, 0x82, 0x45, 0x3C, 0xE3,
    0xFF, 0x02, 0xC5, 0x3A, 0x77, 0x29, 0x61, 0x26,
    0xE5, 0x9E, 0x19, 0x80, 0xCD, 0x49, 0xA5, 0x67,
    0x26, 0xA4, 0x0C, 0xFD, 0xEF, 0x93, 0xA1, 0x81,
    0x41, 0xCF, 0x83, 0x44, 0x2D, 0x0F, 0xDC, 0xDF,
    0x9F, 0x13, 0x51, 0xB2, 0xD0, 0xCF, 0x81, 0x4C,
    0xE9, 0xC7, 0x96, 0x40, 0x2D, 0xC2, 0x21, 0x81,
    0x32, 0xD2, 0x83, 0x60, 0x5B, 0xDD, 0x15, 0x46,
    0x8E, 0xAB, 0xA4, 0xB6, 0xF7, 0x8E, 0x4D, 0xE3,
    0xDE, 0x07, 0x66, 0xFA, 0x99, 0x15, 0xED, 0x28,
    0xE0, 0x0D, 0x90, 0x75, 0x7F, 0x49, 0x49, 0x86,
    0x09, 0x24, 0x77, 0xC9, 0x0C, 0x5F, 0xC3, 0x05,
    0xA5, 0x68, 0x29, 0x08, 0x8D, 0x99, 0x6D, 0x22,
    0x7D, 0x2F, 0x01, 0x8C, 0x1A, 0x16, 0x37, 0x7B,
    0x00, 0x14, 0xA8, 0x18, 0x3F, 0x59, 0xCF, 0x88,
    0x71, 0xC4, 0x65, 0x91, 0x32, 0xBD, 0xDB, 0xA7,
    0x9E, 0x86, 0x9A, 0xE8, 0xF6, 0x5C, 0x93, 0x60,
    0x8D, 0x17, 0x9A, 0x07, 0xD7, 0xD9, 0x94, 0xE0,
    0x58, 0xE5, 0xF5, 0x1B, 0x47, 0xC7, 0x20, 0x9A,
    0x25, 0x86, 0x4D, 0xA9, 0xF1, 0x37, 0x7C, 0x16,
    0xB1, 0xC0, 0x9C, 0x85, 0xB6, 0x6C, 0xC3, 0xD5,
    0x27, 0xFA, 0xB3, 0xF6, 0xB2, 0xDF, 0x6D, 0x6B,
    0xEA, 0x15, 0x20, 0x62, 0x98, 0xBA, 0xC3, 0xE2,
    0x93, 0xF1, 0x0E, 0x2E, 0x9B, 0x78, 0x0E, 0xCE,
    0x03, 0x3A, 0x47, 0xCF, 0xC4, 0x51, 0x22, 0x15,
    0x22, 0xBB, 0x70, 0x9E, 0x1B, 0x94, 0xD8, 0xEA,
    0x74, 0x87, 0x24, 0x21, 0x85, 0xD8, 0xF8, 0xFB,
    0x01, 0x3E, 0x9E, 0x10, 0x73, 0x95, 0xD5, 0x3E,
    0x22, 0xC5, 0x55, 0x02, 0xFC, 0x1E, 0x4A, 0x91,
    0x57, 0x66, 0xF3, 0xC3, 0xB4, 0x63, 0xA3, 0xEE,
    0x4C, 0xB6, 0x82, 0x92, 0x6A, 0x0C, 0x4F, 0x87,
    0xCD, 0x86, 0x18, 0x1A, 0xBC, 0x6F, 0xB9, 0x02,
    0xBD, 0x83, 0x31, 0xDE, 0x18, 0xF5, 0x98, 0x20,
    0xC5, 0xD9, 0x67, 0xD7, 0x84, 0xB1, 0xC0, 0x6E,
    0x5A, 0x94, 0xF3, 0x1E, 0xF8, 0x61, 0x1B, 0x54,
    0x5D, 0x2F, 0x1E, 0x18, 0x4C, 0xEA, 0xC3, 0x12
};

static const unsigned char q_256[] = {
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
```

```
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x43
};

/**
 * @brief Print the value of a sp_int
 * @param num: The number to print
 * @return void
*/
void print_sp_int(sp_int *num) {
    int size = sp_unsigned_bin_size(num);
    char *buffer = (char *)malloc(size * 2 + 1);
    if (buffer == NULL) {
        ESP_LOGE("Print mp_int", "Failed to allocate memory for
            buffer");
        return;
    }
    memset(buffer, 0, size * 2 + 1); // Initialize the buffer to
        zeros

    if (sp_toradix(num, buffer, 16) == MP_OKAY) {
        ESP_LOGI("Print mp_int", "mp_int value: %s", buffer);
    } else {
        ESP_LOGE("Print mp_int", "Failed to convert mp_int to
            string");
    }
    free(buffer);
}

/**
 * @brief Compute Large number Modular Exponetiation with hardware
     Y = (G ^ X) mod P
 * @param g: Base
 * @param x: Exponent
 * @param p: Modulus
 * @param y: Result
 * @return 0 on success, -1 on failure
 */
int powmod(sp_int *g, sp_int *x, sp_int *p, sp_int *y) {
    return esp_mp_exptmod(g,x,p,y);
}

/**
 * @brief Compute Large number Modular Exponetiation with known G (
     generator also known as base) and P (large prime also known as
     modulus). Y = (G ^ X) mod P
 * @param seckey: Exponent
 * @param pubkey: Result
 * @return 0 on success, -1 on failure
 */
int g_pow_p(sp_int *seckey, sp_int *pubkey) {
    int ret;
    DECL_MP_INT_SIZE(large_prime, 3072);
```

```
    NEW_MP_INT_SIZE(large_prime, 3072, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(large_prime, 3072);
    sp_read_unsigned_bin(large_prime, p_3072, sizeof(p_3072));

    DECL_MP_INT_SIZE(generator, 3072);
    NEW_MP_INT_SIZE(generator, 3072, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(generator, 3072);
    sp_read_unsigned_bin(generator, g_3072, sizeof(g_3072));
    ret = esp_mp_exptmod(generator,seckey,large_prime,pubkey);
    if(ret != 0) {
        ESP_LOGE("G_POW_P", "Failed to compute g^x mod p");
        ESP_LOGE("G_POW_P", "Error code: %d", ret);
    }
    sp_zero(large_prime);
    sp_zero(generator);
    FREE_MP_INT_SIZE(large_prime, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(generator, NULL, DYNAMIC_TYPE_BIGINT);
    return 0;
}

/**
 * @brief Generate a random number below Q
 * Q is very close to the maximum value for a 256 bit number. It
    might be worth to compare and regenerate in case mod is
    expensive
 * @param result: The random number
 * @return 0 on success, -1 on failure
 */
int rand_q(mp_int *result) {
    int sz = 32;
    unsigned char *block = (unsigned char *)malloc(sz * sizeof(
       unsigned char));
    if (block == NULL) {
        ESP_LOGE("RAND_Q", "Failed to allocate memory for block");
        return -1; // Memory allocation failed
    }

    esp_fill_random(block, sz);
    sp_read_unsigned_bin(result, block, sz);
    // Possible optimization might not clear the random number
    memset(block, 0, sz);
    free(block);

    DECL_MP_INT_SIZE(small_prime, 256);
    NEW_MP_INT_SIZE(small_prime, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(small_prime, 256);
    sp_read_unsigned_bin(small_prime, q_256, sizeof(q_256));

    sp_mod(result, small_prime, result);

    // Clear
    sp_zero(small_prime);
```

```
    FREE_MP_INT_SIZE(small_prime, NULL, DYNAMIC_TYPE_BIGINT);
    return 0;
}


/**
 * @brief Given two mp_ints, calculate their cryptographic hash
    using SHA256.
 * Possible collision. In the python implementation a delimiter (|)
     is used
 * @param a: First element
 * @param b: Second element
 * @param result: The result of the hash
 * @return 0 on success, -1 on failure
 */
int hash(sp_int *a, sp_int *b, sp_int *result) {
    int ret;
    word32 a_size = sp_unsigned_bin_size(a);
    word32 b_size = sp_unsigned_bin_size(b);
    word32 tmp_size = a_size + b_size;

    byte *tmp = (byte *)malloc(tmp_size);
    if (tmp == NULL) {
        ESP_LOGE("HASH_ELEMS", "Failed to allocate memory for tmp")
            ;
        return -1; // Return an error code
    }
    // Concatenate the two mp_ints
    ret = sp_to_unsigned_bin(a, tmp);
    ret = sp_to_unsigned_bin_at_pos(a_size,b, tmp);

    // Conveniencefunction. Handles Initialisation, Update and
       Finalisation
    if ((ret = wc_Sha256Hash(tmp, tmp_size, tmp)) != 0) {
        WOLFSSL_MSG("Hashing Failed");
        return ret;
    }

    ret = sp_read_unsigned_bin(result, tmp, WC_SHA256_DIGEST_SIZE);
    memset(tmp, 0, tmp_size);
    free(tmp);
    return ret;
}


/**
 * @brief Computes a single coordinate value of the election
    polynomial used for sharing
 * @param exponent_modifier: Unique modifier (usually sequence
    order) for exponent [0, Q]
 * @param polynomial: Election polynomial
 * @param coordinate: The computed coordinate
```

```
 * @return 0 on success, -1 on failure
 */
/*
int compute_polynomial_coordinate(int *exponent_modifier,
    ElectionPolynomial polynomial, sp_int *coordinate) {
    DECL_MP_INT_SIZE(exponent, 256);
    NEW_MP_INT_SIZE(exponent, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(exponent, 256);

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

    DECL_MP_INT_SIZE(small_prime, 256);
    NEW_MP_INT_SIZE(small_prime, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(small_prime, 256);
    sp_read_unsigned_bin(small_prime, q_256, sizeof(q_256));

    for (size_t i = 0; i < polynomial->num_coefficients; i++)
    {
        // Accelerated
        powmod(exponent_modifier, i, small_prime, exponent);
        esp_mp_mulmod(polynomial->coefficients[i].value, exponent,
            small_prime, factor);
        // Not accelerated
        sp_addmod(coordinate, factor, small_prime, coordinate);

        // Reset exponent and factor for the next iteration
        sp_zero(exponent);
        sp_zero(factor);
    }

    // Free
    sp_zero(small_prime);
    FREE_MP_INT_SIZE(small_prime, NULL, DYNAMIC_TYPE_BIGINT);
    sp_zero(exponent)
    FREE_MP_INT_SIZE(exponent, NULL, DYNAMIC_TYPE_BIGINT);
    sp_zero(factor)
    FREE_MP_INT_SIZE(factor, NULL, DYNAMIC_TYPE_BIGINT);
} */

/**
 * @brief Get a hash-based message authentication code(hmac) digest
 * @param key: key (key) in bytes
 * @return error_code
 */
int get_hmac(byte key) {
    /*
    Hmac hmac;
    if(wc_HmacSetKey(&hmac, WC_SHA256, key, sizeof(key)) != 0) {
        ESP_LOGE("HMAC", "Failed to initialise hmac");
        wc_HmacFree(&hmac);
```

```
        return -1;
    }
    if(wc_HmacUpdate(&hmac, data, sizeof(data)) != 0) {
        ESP_LOGE("HMAC", "Failed to update data");
        wc_HmacFree(&hmac);
        return -1;
    }
    if(wc_HmacFinal(&hmac, data_key) != 0) {
        ESP_LOGE("HMAC", "Failed to compute hash");
        wc_HmacFree(&hmac);
        return -1;
    }
    wc_HmacFree(&hmac);
    */
    return 0;
}
void int_to_bytes(int value, uint8_t *bytes) {
    for (int i = 0; i < 4; i++) {
        bytes[3 - i] = (value >> (i * 8)) & 0xFF;
    }
}

/**
 * @brief Key Based Derivitive function in counter mode based on
    NIST SP 800-108 using HMAC as PRF
 * @param key: Session key
 * @param message: Encryption seed. Returns encrypted message after
     operation
 * @return error_code
 */
int kdf(sp_int *key, sp_int *message, sp_int *keystream) {
    /*
     Hmac hmac;
    byte hash[SHA256_DIGEST_SIZE];

    int bits = sp_count_bits(message);
    uint8_t bits_length[4];
    int_to_bytes(bits, bits_length);

    sp_copy(message, keystream);

    int counter = bits / 256;
    uint8_t counter_bytes[4];

    if(wc_HmacSetKey(&hmac, WC_SHA256, key, sizeof(key)) != 0) {
        ESP_LOGE("HMAC", "Failed to initialise hmac");
        wc_HmacFree(&hmac);
        return -1;
    }

    for(int i=0, i < counter, i++) {
        // Convert counter into bytes representation
```

```
        int_to_bytes(i, counter_bytes);

        //Update HMAC with counter
        if(wc_HmacUpdate(&hmac, counter_bytes, sizeof(counter_bytes
            )) != 0) {
            ESP_LOGE("HMAC", "Failed to update data");
            wc_HmacFree(&hmac);
            return -1;
        }
        if(wc_HmacUpdate(&hmac, message, sizeof(message)) != 0) {
            ESP_LOGE("HMAC", "Failed to update data");
            wc_HmacFree(&hmac);
            return -1;
        }
        if(wc_HmacUpdate(&hmac, bits_length, sizeof(bits_length))
            != 0) {
            ESP_LOGE("HMAC", "Failed to update data");
            wc_HmacFree(&hmac);
            return -1;
        }
        if (wc_HmacFinal(&hmac, hash) != 0) {
            ESP_LOGE("HMAC", "Failed to finalize hmac");
            wc_HmacFree(&hmac);
        return -1;
        }

        for (int j = 0; j < SHA256_DIGEST_SIZE; j++) {
            keystream[i * SHA256_DIGEST_SIZE + j] ^= hash[j];
        }
    }

    wc_HmacFree(&hmac);
    */

    return 0;
}


/**
 * Encrypts a variable length message with a given random nonce and
     an ElGamal public key.
 * @param coordinate: message (m) to encrypt; must be in bytes.
 * @param nonce: Randomly chosen nonce in [1, Q).
 * @param public_key: ElGamal public key.
 * @param encryption_seed: Encryption seed (Q) for election.
 * @param encrypted_coordinate: The encrypted message.
 */
int hashed_elgamal_encrypt(sp_int *coordinate, sp_int *nonce,
    sp_int *public_key, sp_int *seed, sp_int *encrypted_coordinate)
    {
    //DECL_MP_INT_SIZE(large_prime, 3072);
```

```
    //NEW_MP_INT_SIZE(large_prime, 3072, NULL, DYNAMIC_TYPE_BIGINT)
        ;
    //INIT_MP_INT_SIZE(large_prime, 3072);
    //sp_read_unsigned_bin(large_prime, p_3072, sizeof(p_3072));
    // DECL pad, pubkey_pow_n

    //g_pow_p(nonce, pad); //alpha
    //powmod(public_key, nonce, large_prime ,pubkey_pow_n); //beta
    //hash(pad, pubkey_pow_n, session_key); //secret_key

    //size_t pad the coordinates
    //int len = sp_unsigned_bin_size(coordinate);
    //int padded_size = len + (BLOCK_SIZE - (len % BLOCK_SIZE));
    //byte padded_coordinate[padded_size];
    //memset(padded_coordinate, 0, padded_size);
    //sp_to_unsigned_bin_at_pos(padded_size - len, coordinate,
        padded_coordinate);

    //kdf(session_key, encryption_seed, data_key); // KDF in
        counter mode
    /*
    mac_key = get_hmac(
        session_key.to_hex_bytes(), encryption_seed.to_hex_bytes(),
            bit_length
    )
    to_mac = pad.to_hex_bytes() + data
    mac = get_hmac(mac_key, to_mac)

    */
    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() {
    //compute_polynomial_coordinate(receiver_sequence_order,
        sender_guardian_polynomial, coordinate);
    //rand_q(nonce);
    //hash(receiver_owner_id, receiver_sequence_order, seed);
    //hashed_elgamal_encrypt(coordinate, nonce,
        receiver_guardian_public_key.key, seed, encrypted_coordinate
        );
```

```
    //return PartialKeyBackup(sender_guardian_id, receiver_owner_id
        , receiver_sequence_order, encrypted_coordinate)
    return 0;
}

/**
 * @brief Generates a polynomial for sharing election keys
 * @param coefficients:  Number of coefficients of polynomial
 * @param Polynomial used to share election keys. Contains value,
    commitment, and proof
 * @return 0 on success, -1 on failure
 */
int generate_polynomial(ElectionPolynomial *polynomial) {
    SchnorrProof proof;
    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(value, 256);
    NEW_MP_INT_SIZE(value, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(value, 256);

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

    for (int i = 0; i < polynomial->num_coefficients; i++) {
        rand_q(value);
        rand_q(nonce);
        g_pow_p(value, commitment);
        polynomial->coefficients[i].value = (sp_int*)XMALLOC(
            MP_INT_SIZEOF(MP_BITS_CNT(256)), NULL,
            DYNAMIC_TYPE_BIGINT);
        polynomial->coefficients[i].commitment = (sp_int*)XMALLOC(
            MP_INT_SIZEOF(MP_BITS_CNT(3072)), NULL,
            DYNAMIC_TYPE_BIGINT);

        if (polynomial->coefficients[i].value) {
            XMEMSET(polynomial->coefficients[i].value, 0,
                MP_INT_SIZEOF(MP_BITS_CNT(256)));
            mp_init_size(polynomial->coefficients[i].value,
                MP_BITS_CNT(256));
        }
        if (polynomial->coefficients[i].commitment) {
            XMEMSET(polynomial->coefficients[i].commitment, 0,
                MP_INT_SIZEOF(MP_BITS_CNT(3072)));
            mp_init_size(polynomial->coefficients[i].commitment,
                MP_BITS_CNT(3072));
        }

        sp_copy(nonce, polynomial->coefficients[i].value);
```

```
        sp_copy(commitment, polynomial->coefficients[i].commitment)
            ;

        make_schnorr_proof(nonce, commitment, nonce, &proof);
        polynomial->coefficients[i].proof = proof;
    }

    sp_zero(nonce);
    sp_zero(value);
    sp_zero(commitment);
    FREE_MP_INT_SIZE(nonce, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(value, NULL, DYNAMIC_TYPE_BIGINT);
    FREE_MP_INT_SIZE(commitment, NULL, DYNAMIC_TYPE_BIGINT);
    return 0;
}
/**
 * @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, ElGamalKeyPair *key_pair
    ) {
    ElectionPolynomial polynomial;
    polynomial.num_coefficients = quorum;
    polynomial.coefficients = (Coefficient*)XMALLOC(quorum * sizeof
        (Coefficient), NULL, DYNAMIC_TYPE_BIGINT);
    if (polynomial.coefficients == NULL) {
        ESP_LOGE("Generate Election Key Pair", "Failed to allocate
            memory for coefficients");
        return -1;
    }
    generate_polynomial(&polynomial);
    sp_copy(polynomial.coefficients[0].value, key_pair->secret_key)
        ;
    sp_copy(polynomial.coefficients[0].commitment, key_pair->
        public_key);
    return 0;
}


/**
 * @brief Given an ElGamal keypair and a nonce, generates a proof
     that the prover knows the secret key without revealing it.
 * @param seckey: The secret key
 * @param pubkey: The public key
 * @param nonce: A random element in [0,Q)
 * @param proof: The Schnorr proof
 * @return 0 on success, -1 on failure
 */
```

```
int make_schnorr_proof(sp_int *seckey, sp_int *pubkey, sp_int *
    nonce, SchnorrProof *proof) {
    proof->pubkey = (sp_int*)XMALLOC(MP_INT_SIZEOF(MP_BITS_CNT
        (3072)), NULL, DYNAMIC_TYPE_BIGINT);
    proof->commitment = (sp_int*)XMALLOC(MP_INT_SIZEOF(MP_BITS_CNT
        (3072)), NULL, DYNAMIC_TYPE_BIGINT);
    proof->challenge = (sp_int*)XMALLOC(MP_INT_SIZEOF(MP_BITS_CNT
        (256)), NULL, DYNAMIC_TYPE_BIGINT);
    proof->response = (sp_int*)XMALLOC(MP_INT_SIZEOF(MP_BITS_CNT
        (4096)), NULL, DYNAMIC_TYPE_BIGINT);

    if (proof->pubkey != NULL) {
        XMEMSET(proof->pubkey, 0, MP_INT_SIZEOF(MP_BITS_CNT(3072)))
            ;
    }
    if (proof->commitment != NULL) {
        XMEMSET(proof->commitment, 0, MP_INT_SIZEOF(MP_BITS_CNT
            (3072)));
    }
    if (proof->challenge != NULL) {
        XMEMSET(proof->challenge, 0, MP_INT_SIZEOF(MP_BITS_CNT(256)
            ));
    }
    if (proof->response != NULL) {
        XMEMSET(proof->response, 0, MP_INT_SIZEOF(MP_BITS_CNT(4096)
            ));
    }

    mp_init_size(proof->pubkey, MP_BITS_CNT(3072));
    mp_init_size(proof->commitment, MP_BITS_CNT(3072));
    mp_init_size(proof->challenge, MP_BITS_CNT(256));
    mp_init_size(proof->response, MP_BITS_CNT(4096));

    sp_copy(pubkey, proof->pubkey);
    g_pow_p(nonce, proof->commitment);
    hash(pubkey, proof->commitment, proof->challenge);

    // a + bc ^ q = nonce + seckey * challenge ^ q
    DECL_MP_INT_SIZE(q, 256);
    NEW_MP_INT_SIZE(q, 256, NULL, DYNAMIC_TYPE_BIGINT);
    INIT_MP_INT_SIZE(q, 256);
    sp_read_unsigned_bin(q, q_256, sizeof(q_256));

    sp_mul(seckey,proof->challenge,proof->response);
    sp_addmod(nonce,proof->response,q,proof->response);

    sp_zero(q);
    FREE_MP_INT_SIZE(q, NULL, DYNAMIC_TYPE_BIGINT);
    return 0;
}
```

A.3 View Component

```
#include <stdio.h>
#include "view.h"

void func(void)
{

}
```

A.4 Adapter Component

```
#include <stdio.h>
#include "adapter.h"

void func(void)
{

}
```